Aljoscha Kirchner



Entwicklung von Methoden zur abstrakten Modellierung von Automotive Systems-on-Chips





Entwicklung von Methoden zur abstrakten Modellierung von Automotive Systems-on-Chips

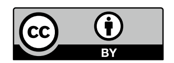
Aljoscha Kirchner

Entwicklung von Methoden zur abstrakten Modellierung von Automotive Systems-on-Chips



Aljoscha Kirchner
Tübingen, Deutschland

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.



This work has been developed in the ITEA3 project COMPACT. COMPACT is funded by the Finnish funding agency for innovation Tekes (Diary Number 3089/31/2017) and the German ministry of education and research (BMBF) (reference number: 01/S17028). The authors are responsible for the content of this publication.

ISBN 978-3-658-38436-4 ISBN 978-3-658-38437-1 (eBook)
https://doi.org/10.1007/978-3-658-38437-1

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Planung/Lektorat: Stefanie Eggert
Springer Vieweg ist ein Imprint der eingetragenen Gesellschaft Springer Fachmedien Wiesbaden GmbH und ist ein Teil von Springer Nature.
Die Anschrift der Gesellschaft ist: Abraham-Lincoln-Str. 46, 65189 Wiesbaden, Germany

*Ich widme diese Arbeit meiner Frau.
Ohne sie wäre ich nur ein Teil meines
Selbst.*

Danksagung

Die in dieser Dissertation vorgestellten Arbeiten entstanden während meiner Tätigkeit als Doktorand bei der *Robert Bosch GmbH* in Reutlingen in den Jahren von 2017 bis 2021. An dieser Stelle möchte ich allen danken, die zum Gelingen dieser Arbeit und damit zum Entstehen dieses Buches beigetragen haben.

Mein Dank gilt **Prof. Dr. Oliver Bringmann**, welcher bei unseren zahlreichen Treffen durch seine wertvolle Anleitung und in Diskussionen den fachlichen Rahmen für diese Arbeit gegeben hat. Darüber hinaus möchte ich mich bei **Prof. Dr. Joachim Gerlach** für die Übernahme des Koreferats bedanken. Mein ganz besonderer Dank gilt **Dr. Jan-Hendrik Oetjens**, welcher das Gelingen dieser Arbeit durch sein stets offenes Ohr, seine fachliche Unterstützung und seine zahlreichen Ratschläge ermöglicht hat. Auch möchte ich an dieser Stelle **Andreas Mauderer** danken, der mir besonders zu Beginn meiner Doktorandenzeit fachlich wie auch organisatorisch stets mit Rat zur Seite stand.

Einen wichtigen Beitrag zu dieser Arbeit leisteten darüber hinaus die von mir betreuten Masterstudenten **Markus Friedrich**, **Sascha Lehmann** und **Chukwuma Onuoha Onuoha,** welche über die fachliche Leistung hinaus meine Arbeit durch unsere freundschaftliche Zusammenarbeit bereicherten.

Bedanken möchte ich mich auch bei den Kollegen **Bernhard Opitz**, **Rafał Baranowski**, **Michael Veith**, **Max Geer**, **Viktor Schrull** sowie dem gesamten **Expert Domain Concept Team** der *Robert Bosch GmbH*, die durch ihre fachlichen wie auch privaten Diskussionen und Gespräche meine Zeit als Doktorand bereicherten und zum Gelingen der Arbeit beitrugen.

Ein besonderer Dank gilt zudem **meiner ganzen Familie,** auf deren Unterstützung und Rückhalt ich stets zählen kann und die mir so maßgeblich die Kraft für mein Vorhaben gab.

Zum Schluss möchte ich von ganzem Herzen **meiner Frau** danken. Ohne ihre grenzenlose Unterstützung und die Bereitschaft, auch ihr Leben auf das Gelingen dieser Arbeit auszurichten, hätte ich mein Ziel nicht erreichen können.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis



1 Einleitung

Es sind die großen Technologietrends der heutigen Zeit, welche die Automobilbranche nach wie vor zu einem der großen Treiber der Innovationsentwicklung werden lässt. Dabei ist es neben der Digitalisierung und der Elektrifizierung des Automobils insbesondere die Entwicklung des autonomen Fahrens, welche die Komplexität der Anforderungen stetig steigen lässt. Aus den steigenden Anforderungen in der Automobilbranche resultieren steigende Anforderungen bei den Zulieferern. So benötigten beispielsweise die heute fast standardmäßig verbauten Fahrassistenzsysteme ein hohes Maß an Vernetzung und eine Vielzahl an gemessenen Umgebungsparametern. Um diese Messwerte in Echtzeit verarbeiten und bereitstellen zu können, bedarf es einer großen Menge hochleistungsfähiger Automotive Systems-on-Chips (SoC). Zum einen steigen somit die technischen Anforderungen, da eine immer größere Menge an Daten immer schneller verarbeitet werden muss, zum anderen die nichtfunktionalen Anforderungen an die SoC-Entwicklung, wie Entwicklungszeit und Teilekosten, um in der aktuellen Konkurrenzsituation bestehen zu können.

Um diese technischen und nichtfunktionalen Anforderungen auch zukünftig erfüllen zu können, bedarf es einer fortlaufenden Effizienzsteigerung der SoC-Entwicklung. Die Effizienz im Sinne der Wirtschaftlichkeit beschreibt das Verhältnis zwischen aufgebrachtem Aufwand und dem daraus erhaltenen Ergebnis. In der SoC-Entwicklung lässt sich der Begriff des Ergebnisses auf die Qualität des erhaltenen Endproduktes herunterbrechen. Die Qualität des SoCs im Sinne von z. B. Zuverlässigkeit, Robustheit, Fehlerfreiheit, Verarbeitungsgeschwindigkeit ist ein entscheidendes Bewertungskriterium für die Entwicklung.

Der aufgebrachte Aufwand, um diese Qualität zu erreichen, lässt sich sowohl in Kosten als auch in Zeit bemessen. Diese drei Parameter Kosten, Zeit und Qualität, deren Verhältnis in dieser Arbeit für die Effizienz der SoC-Entwicklung

A. Kirchner, *Entwicklung von Methoden zur abstrakten Modellierung von Automotive Systems-on-Chips*, https://doi.org/10.1007/978-3-658-38437-1_1

stehen, werden in der Literatur im „Magischen Dreieck“, auch Entwurfsdreieck genannt, abgebildet. Die Literatur spricht deswegen von einem „Magischen Dreieck“, weil sich die Veränderung einer der Parameter Kosten, Zeit und Qualität in einem bestehenden Entwicklungs-Flow auf mindestens einen der jeweils anderen Parameter auswirkt. Dies soll am nachfolgenden Beispiel aus der SoC-Entwicklung veranschaulicht werden.

Die erfolgreiche Markteinführung eines neuen Produktes hängt für den Automobilhersteller aufgrund der Konkurrenzsituation meist stark von dem Zeitpunkt der Markteinführung ab. Aufgrund der in der Regel kunden- und anwendungsspezifischen Entwicklung eines Automotive SoCs überträgt sich dieser Termindruck auf den SoC-Hersteller. Kommt es nun im Zuge der SoC-Entwicklung aufgrund einer unzureichenden Analyse der Kundenanforderungen beispielsweise zu Verzögerungen, entsteht ein Zielkonflikt. Um den Liefertermin trotz Verzögerung einhalten zu können, gibt es in diesem abstrahierten Beispiel zwei Möglichkeiten:

- Es müssen mehr Entwickler/-innen hinzugezogen werden oder die vorhandenen Entwickler/-innen müssen mehr Stunden an dem SoC-Projekt arbeiten. Dadurch jedoch **steigen die Kosten** der Entwicklung und damit des Endproduktes. Darüber hinaus lässt sich die Entwicklungszeit nur bis zu einem gewissen Punkt durch Hinzuziehen weiterer Entwickler/-innen reduzieren, da durch jeden weiteren Entwickler und durch jede weitere Entwicklerin der benötigte Aufwand für Kommunikation und Koordination im SoC-Projekt steigt [1].
- Alternativ kann Zeit eingespart werden, beispielsweise durch eine Minimierung des Test- und Verifikationsumfangs. Somit steigt jedoch die Gefahr einer **Minderung der Qualität** des Endproduktes bzw. für Nacharbeiten am SoC, welche wiederum Zeit und Kosten verursachen. Die Senkung der Qualität ist in der Entwicklung von Automotive SoCs in der Regel aufgrund der hohen Sicherheitsanforderungen an diese Systeme kein mögliches Szenario.

Die hier gezeigte Verknüpfung der Parameter Kosten, Zeit und Qualität im „Magischen Dreieck“ verdeutlicht, bezogen auf die Automotive-SoC-Entwicklung, die Hürde bei der Effizienzsteigerung. Eine Steigerung der Effizienz als Verhältnis von Aufwand (Zeit und Kosten) zu Ergebnis (Qualität) kann bei gleichbleibender Qualität nur durch eine Senkung des Aufwandes erreicht werden. Jedoch kann durch die im „Magischen Dreieck“ veranschaulichte Abhängigkeit der Parameter eine Reduktion des Zeit- bzw. Kostenaufwandes nur durch eine Steigerung des jeweils anderen Parameters erfolgen. Die Abhängigkeit der Parameter kann auf lange Sicht lediglich durch die Einführung verbesserter Methoden, Arbeitsweisen

und Technologien erfolgen [2, pp. 8–11]. So kann beispielsweise die Anwendung einer neuen Methode, welche den Entwicklungsprozess automatisiert, zu einer Senkung des Aufwandes und gleichzeitig zu einer Steigerung der Qualität des Ergebnisses führen, da durch die Automatisierung die Fehleranfälligkeit im Vergleich zu manuellen Tätigkeiten in der Regel gesenkt werden kann.

1.1 Zielsetzung

Ziel dieser Arbeit ist die **Effizienzsteigerung in der Entwicklung von Automotive System-on-Chips** durch die direkte oder indirekte Senkung der entstehenden Aufwände. Hierzu soll eine Methode entwickelt werden, mit dem Ziel, die bestehenden Defizite des aktuellen Entwicklungs-Flows in der Automotive-SoC-Entwicklung zu minimieren. Ein besonderer Fokus der Arbeit liegt dabei auf dem Erstellen und Pflegen der Spezifikationsdokumente.

Durch die hier entwickelte Methode soll eine Arbeitsweise implementiert werden, welche durch eine ausführliche Analyse der Kundenforderungen und der eindeutigen Beschreibung der Systemanforderungen ein einheitliches Systemverständnis aller „Stakeholder" (Interessenvertreter) schafft. Somit soll die Notwendigkeit von **nachträglichen Änderungen** an den **vereinbarten Anforderungen** zwischen Kunde und Entwickler **verringert** werden und die zusätzlichen Aufwände und Kosten, die dabei entstehen, sollen reduziert werden.

Ziel der Arbeit ist es darüber hinaus, eine Methode zu schaffen, welche die Vollständigkeit und Eindeutigkeit der Spezifikation steigert und dadurch das Auftreten von **Implementierungsfehlern,** die aufgrund einer unzureichenden Spezifikation entstehen, **verringert**. Werden Implementierungsfehler nicht entdeckt, führen sie zu einer Minderung der Qualität. Werden sie entdeckt und müssen behoben werden, entstehen zusätzliche Kosten und es kommt zu Verzögerungen in der Entwicklung. Zusätzlich zu Implementierungsfehlern kommt es immer wieder dazu, dass Entwurfsentscheidungen, welche während der Implementierung getroffen wurden, nicht ausreichend in der Spezifikation dokumentiert werden und hierdurch **Inkonsistenzen** zwischen implementiertem Entwurf und Spezifikation entstehen. Dadurch ergeben sich erhebliche Mehraufwände und damit Kosten, zum Beispiel für die Verifikation des SoCs, da hier das implementierte System gegen die Spezifikation verifiziert wird.

Durch die Einführung von neuen Methoden und Arbeitsweisen in bestehende Arbeitsprozesse kommt es zu Beginn oftmals zu einer Einbuße der Effizienz, zum Beispiel aufgrund von Kompatibilitätsproblemen und fehlenden Schnittstellen zwischen neuen und bereits etablierten Tools. Daher ist eine entscheidende

Anforderung an die hier vorgestellte Methode, dass sie sich in den **aktuellen Automotive-SoC-Entwicklungs-Flow integrieren** lässt. Darüber hinaus verursachen neue Methoden oftmals zusätzliche Arbeit und benötigen eine Umstellung der gewohnten Arbeitsweisen. Um die Effizienz der SoC-Entwicklung nicht zu verringern, sondern zu steigern, ist es wichtig, durch die Anwendung der neuen Methode den **Spezifikationsaufwand** so gering wie möglich zu **halten** bzw. weiter zu **verringern,** verglichen mit der aktuell verwendeten Methode.

Um durch neue Methoden die Effizienz der Entwicklung steigern zu können, ist eine einfache und **anwenderfreundliche** Handhabe der enthaltenen Mechanismen und Arbeitsschritte entscheidend. Lässt sich die Methode nur schwer erlernen und umsetzen, führt dies zu erheblichen Verzögerungen, zusätzlichen Kosten und kann der Qualität des zu entwickelnden Systems schaden.

Neben der Spezifikation besteht ein großer Teil der manuell getätigten Arbeiten in der Implementierung sowie in der bereits angesprochenen Verifikation von SoCs. Dabei liegt ein großes Defizit darin, dass Informationen in einer nicht maschinenlesbaren Sprache spezifiziert sind und daher diese Informationen im nächsten Schritt manuell in Programmiersprachen oder sonstige Formate übersetzt werden müssen. Ziel der Methode ist es daher, den **manuellen Aufwand** beim **Systementwurf** und der **Verifikation** zu **reduzieren** und somit die benötigte Zeit und damit die Kosten für manuelle Arbeitsschritte zu verringern.

Das Ziel der in dieser Arbeit vorgestellten modellbasierten Entwicklungsmethode ist die **Steigerung der Entwicklungseffizienz** heutiger **Automotive-Systems-on-Chip-Entwicklungen**. Um dieses Ziel zu erreichen, wurden folgende Anforderungen an die Methode analysiert:

A1 Durch Anwendung der Methode soll die Notwendigkeit von **nachträglichen Änderungen** an den **vereinbarten Anforderungen** zwischen Kunde und Entwickler **verringert** werden.

A2 Die Methode soll eine **Reduzierung** der aufgrund unzureichender Eindeutigkeit der Spezifikation entstandenen **Implementierungsfehler** ermöglichen.

A3 **Inkonsistenzen** zwischen implementiertem System und Spezifikation sollen durch die hier vorgestellte Methode **reduziert** werden.

A4 Die Methode soll sich in den **aktuellen Automotive-SoC-Entwicklungs-Flow integrieren** lassen und Schnittstellen zu anderen Beschreibungsarten bieten.

A5 Der **Aufwand bei der Spezifikation** mittels der neuer Methode soll im Vergleich zum aktuellen Vorgehen konstant gehalten oder **vermindert** werden.

A6 Die Methode muss **anwenderfreundlich** sowohl für den Kunden als auch für den SoC-Entwickler sein, um eine Einführung der Methode zu erleichtern und die Fehleranfälligkeit zu reduzieren.

A7 Die Methode soll eine **Reduzierung des Aufwandes** beim **Systementwurf** ermöglichen.

A8 Die Methode soll eine **Reduzierung des Aufwandes** bei der **Verifikation** des Systementwurfs ermöglichen.

1.2 Aufbau der Arbeit

Die vorliegende Arbeit ist wie folgt gegliedert: Kapitel 2 befasst sich mit den Grundlagen, welche für das Verständnis der hier behandelten Arbeit benötigt werden. Der Stand der Technik, bezogen auf die hier zugrundeliegende Thematik, sowie die Abgrenzung gegenüber existierenden Arbeiten werden in Kapitel 3 behandelt. In Kapitel 4 wird die entwickelte modellbasierte Entwicklungsmethode für die Modellierung von Automotive Systems-on-Chips vorgestellt. Anschließend werden auf Grundlage eines industrienahen Beispiels in Kapitel 5 Anwendung und Implementierung der Methode beschrieben. Kapitel 6 befasst sich mit der Evaluierung der modellbasierten Entwicklungsmethode. In Kapitel 7 wird schließlich die Arbeit zusammengefasst und ein Ausblick auf mögliche weiterführende Arbeiten gegeben.



2 Grundlagen

Im folgenden Kapitel werden die zum Verständnis dieser Arbeit benötigten Grundlagen aus den Bereichen der SoC-Entwicklung und der Modellierung behandelt. Als Teil dessen, erfolgt darüber hinaus eine Definition der in dieser Arbeit verwendeten Begriffe. Zu Beginn wird in Abschnitt 2.1 auf die Besonderheiten der Automobilindustrie sowie auf die Auswirkungen, die diese auf die SoC-Entwicklung haben, eingegangen. In dem darauffolgenden Abschnitt 2.2 werden Grundlagen und Begrifflichkeiten Eingebetteter Systeme sowie SoCs behandelt, um anschließend in Abschnitt 2.3 auf die Phasen der SoC-Entwicklung näher einzugehen. Abschnitt 2.4 behandelt auf Grundlage einer Mehrfachfallstudie aktuelle Defizite im Bereich des Anforderungsmanagements und der Kommunikation in der Automobilbranche. Grundlagen und Modellierungssprachen der modellbasierten Systementwicklung werden in Abschnitt 2.5 beschrieben, bevor Abschnitt 2.6 auf relevante Themen der SoC-Entwicklungsmethodik eingeht. Zum Ende des Kapitels folgt in Abschnitt 2.6.8 eine Beschreibung des Ansatzes des modellgetriebenen Systementwurfs.

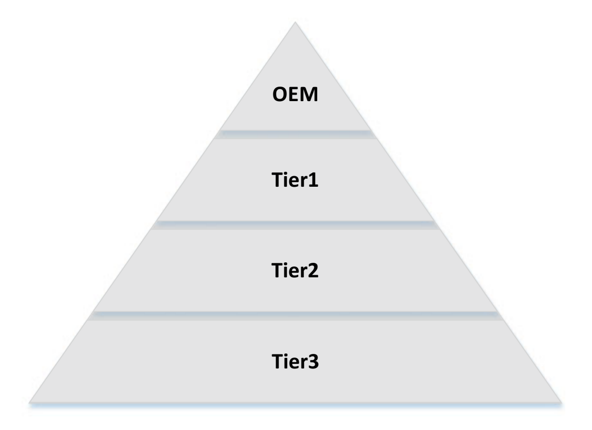
2.1 Automobilindustrie im Wandel

In den Anfängen der Automobilindustrie fertigten die Automobilhersteller, genannt „Original Equipment Manufacturer“ (OEM, synonym zu Fahrzeughersteller), ihre Automobile zu großen Teilen noch selbst. Doch durch die stetig steigende Komplexität waren die Automobilhersteller gezwungen, Kooperationen mit sogenannten Zulieferern einzugehen und Technologieführerschaften abzugeben. Hierdurch wurden aus den Automobilherstellern über die Jahre Koordinatoren einer Vielzahl an Zulieferern. Dennoch behielten die OEMs eine

A. Kirchner, *Entwicklung von Methoden zur abstrakten Modellierung von Automotive Systems-on-Chips*, https://doi.org/10.1007/978-3-658-38437-1_2

hohe Marktmacht und so prägt besonders die Automobilindustrie heute die Pyramidenstruktur der Zulieferkette, wie in Abbildung 2.1 dargestellt [3].

Abbildung 2.1
Zuliefererpyramide [4]



An der Spitze der Pyramide stehen die OEMs. Diese entwickeln, produzieren und vermarkten das Auto als Gesamtsystem. Dabei liegt ein Schwerpunkt der heutigen Produktion durch die OEMs in der Kombination selbst produzierter oder von Zulieferern bezogener Teilsysteme und Komponenten. An zweiter Stelle der Zuliefererpyramide stehen die Tier1 (abgeleitet vom Englischen tier, Schicht). Diese entwickeln und liefern komplexe Teilsysteme, wie zum Beispiel Fahrassistenzsysteme, und verfügen in ihrem Bereich meist über eine Technologieführerschaft. Tier1-Zulieferer wiederum fertigen die entwickelten Teilsysteme vollständig selbst oder kombinieren sie aus Komponenten, welche sie wiederum von Tier2-Zulieferern beziehen. Am Sockel der Pyramide steht Tier3, welcher einzelne Komponenten sowie Ersatzteile herstellt. Der Anteil der Wertschöpfung sinkt von der Spitze der Pyramide zum Sockel hin. Jedoch führen aktuelle Trends in der Automobilindustrie zu einer Verschiebung der Wertschöpfungsanteile und damit zur Verschiebung der Marktmacht hin zu den Zulieferern [4].

So entwickelt sich das Automobil immer mehr vom reinen Nutzwert als Beförderungsmittel zum Erlebniswert. Heute beinhalten Fahrzeuge vollvernetzte Entertainment- sowie Infotainmentsysteme und unterstützen zudem den Fahrer im Straßenverkehr mittels einer Vielzahl an Assistenzsystemen. Durch die damit einhergehende Digitalisierung verlagern sich die Innovationskraft, die Gewinnprognosen und damit auch die Marktmacht immer stärker weg von traditionellen Geschäftsbereichen und hin zu Geschäftsbereichen wie digitale Dienstleistungen (Digital services) und Software, wie in Abbildung 2.2 veranschaulicht [5].



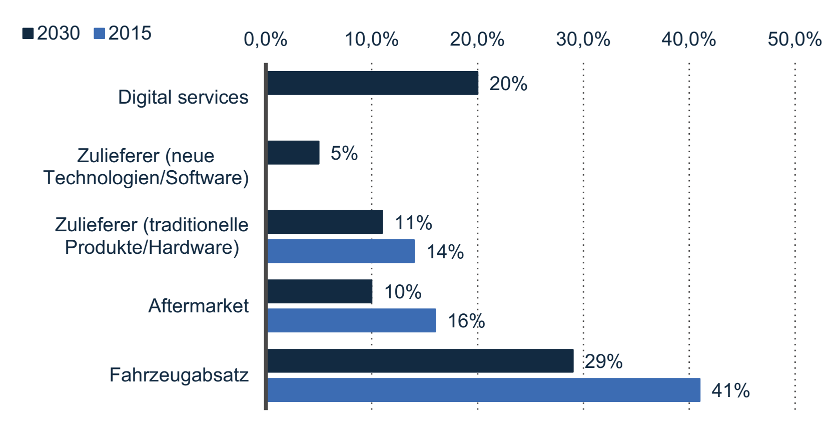


Abbildung 2.2 Verteilung des Gewinns in der weltweiten Automobilindustrie [6]

Das hier gezeigte Diagramm illustriert den Gewinn und unterteilt diesen dabei in fünf Geschäftsbereiche: Fahrzeugabsatz, Aftermarketing, Zulieferer neuer Technologien und Software, Zulieferer traditioneller Technologien und Hardware, Digital services. Die hier nicht abgebildeten Gewinne verteilen sich vorrangig auf Versicherungen und Finanzierungen. In Hellblau sind die Gewinnzahlen der weltweiten Automobilhersteller im Jahr 2015 dargestellt. Wie zu sehen, liegt der Hauptanteil von 41 % im Geschäftsbereich Fahrzeugabsatz, während der Bereich Aftermarket bei 16 % und der Gewinn der Zulieferer traditioneller Technologien bei 14 % liegt. Die Bereiche Zulieferer neuer Technologien und Digital services lagen im Jahr 2015 noch bei etwa 0 %. Die in Dunkelblau dargestellte Prognose der Firma GP Bullhound für das Jahr 2030 zeigt klar den aktuellen und zukünftigen Trend in der Automobilindustrie. So sinkt die Prognose für den Gewinn durch Fahrzeugabsatz, Aftermarket und Zulieferer traditioneller Technologien um insgesamt 21 % und verschiebt sich hin zum Bereich der Zulieferer neuer Technologien und Digital services. Aufgrund der aktuellen Neukonfiguration der Automobilbranche und der damit gezeigten Verschiebung der Wertschöpfungsanteile wächst der Druck auf allen Ebenen der Zuliefererpyramide, sich an die fortschreitenden Veränderungen anzupassen.

Parallel dazu stellt sich heute in vielen der großen Absatzmärkte der Vergangenheit eine Sättigung des Automobilmarktes ein. Dabei dient die Anzahl der

Pkws pro 1.000 Einwohner, genannt Pkw-Dichte, als ein wichtiger Indikator für die Sättigung [7]. Abbildung 2.3 vergleicht die Pkw-Dichte der Eurozone, Japans, Chinas und Indiens im Jahr 2015.

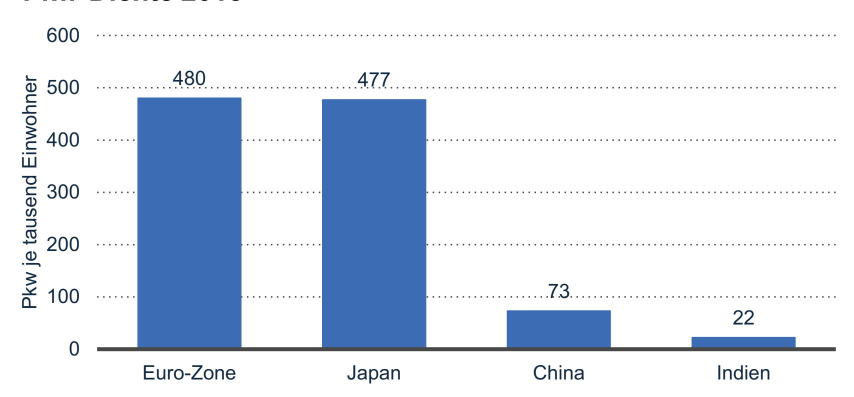


Abbildung 2.3 Pkw-Dichte 2015 [8]

Sowohl in der Eurozone als auch in Japan kommen auf 1.000 Einwohner knapp 500 Pkws. Das bedeutet, jede zweite Person über 18 Jahre besitzt im Schnitt ein Auto. Hierbei spricht man vom Eintreten einer Sättigung. Vergleicht man hierzu die Dichte von China mit 73 Pkws und Indien mit lediglich 22 Pkws pro 1.000 Einwohner, wird schnell das Potenzial solcher Märkte deutlich. Dementgegen steigt in den gesättigten Ländern die Wettbewerbsintensität, da Marktanteile im Wesentlichen nur noch durch Verdrängung anderer Konkurrenten gewonnen werden können [7].

Um sich in diesem Wettbewerb dennoch behaupten zu können und den steigenden Kundenanforderungen gerecht zu werden, setzen viele Hersteller auf die Erweiterung des Modell- und Variantenangebots. Ein Beispiel hierfür ist die Modellportfolioerweiterung von Mercedes-Benz zwischen den Jahren 1990 und 2009 von 5 auf 16 verschiedene Klassen bzw. Baureihen, von denen es jeweils mehrere Varianten gibt [9]. Hinzu kommen – im Zeitraum zwischen der Entwicklung neuer Generationen – regelmäßige „Faceliftings" (Modellpflege) der Pkw-Modelle. Um diese hohe Angebotsvielzahl zu ermöglichen und weiterhin wettbewerbsfähig zu bleiben, müssen die Entwicklungszeiten der Modelle reduziert werden. Dies hat wiederum zwei Effekte auf die Zuliefererpyramide: Zum

einen sind die OEMs gezwungen, die Entwicklung der Teilsysteme und Komponenten verstärkt auf die Zulieferer zu verlagern, um so Kosten und Zeitaufwand reduzieren zu können, und zum anderen verlagert sich der Zeitdruck durch eine strenge terminliche Zielsetzung der OEMs gleichermaßen auf die Zulieferer [10]. Somit sehen sich auch die SoC-Entwickler der Herausforderung gegenüber, die Entwicklungszeiten trotz steigender Komplexität der SoCs weiter zu verringern, um auch in Zukunft konkurrenzfähig zu bleiben [9].

2.2 Eingebettete Systeme

Spätestens seit der Digitalisierung und Vernetzung unseres urbanen Lebensraums sind Eingebettete Systeme nicht immer sichtbarer, jedoch oftmals funktionsrelevanter Bestandteil unseres Alltags. Unter Eingebetteten Systemen versteht man elektronische Rechensysteme, welche als integraler Bestandteil eines übergeordneten Systems eine vordefinierte Aufgabe erfüllen. Dabei erfüllen Eingebettete Systeme eine große Bandbreite von Funktionen und werden in nahezu jedem Wirtschaftszweig und Bereich unseres Lebens eingesetzt. Beispiele für Einsatzmöglichkeiten aus dem Automobilbereich sind:

- Temperatur- und Klimaregelung
- Fahrassistenzsysteme (z. B. Einparkhilfe, Verkehrszeichenerkennung)
- Fahrsicherheitssysteme (z. B. Antiblockiersystem, Notbremssystem)
- Multimediasysteme
- Navigationssysteme
- Automatisiertes Fahren

Eingebettete Systeme lassen sich nach Schaltkreistechnologie und der damit verbundenen Konfigurierbarkeit unterteilen. Dabei gibt es die Oberkategorien der herstellerkonfigurierten und der anwenderkonfigurierbaren Schaltkreise. Es gibt pro Oberkategorie je zwei Arten von Schaltkreisen:

- **Wiederprogrammierbare Schaltkreise,** wie beispielsweise Field Programmable Gate Arrays (FPGAs), lassen sich beliebig oft neu programmieren bzw. rekonfigurieren und somit an die jeweilige Anwendung anpassen.
- **Einmalprogrammierbare Schaltkreise** lassen sich einmal zu Beginn durch den Kunden konfigurieren. Dazu werden je nach Verfahren durch Programmierspannung/-strom die vorhandenen Isolationen bzw. Verbindungen zwischen den Logikgattern irreversibel manipuliert.

- **Teilweise kundenspezifische Schaltkreise** werden mittels vorgefertigter Masken durch den Hersteller konfiguriert. Die Zusammensetzung der Blöcke in der Makrozelle geschieht dabei nach Wunsch des Kunden; die Blöcke selbst sind durch den Hersteller definiert.
- **Vollkundenspezifische Schaltkreise** werden genau nach den Anforderungen des Kunden und in der Regel anwendungsspezifisch konfiguriert.

Die verschiedenen Schaltkreistechnologien haben einen direkten Einfluss auf die technischen Eigenschaften des Eingebetteten Systems und damit auf die benötigten Zeit- und Kostenaufwände in der Entwicklung. Da der Fokus dieser Arbeit auf der Entwicklung von SoCs liegt, soll in der nachfolgenden Tabelle 2.1 die Auswirkung der Schaltkreistechnologie anhand des Vergleichs zwischen SoC als Beispiel eines vollkundenspezifischen Schaltkreises und FPGA als Beispiel eines wiederprogrammierbaren Schaltkreises dargestellt werden. Hierbei handelt es sich um einen ungefähr qualitativen Vergleich [11].

Tabelle 2.1 Vergleich SoC – FPGA [11, p. 25]

	Kundenspezifischer SoC	**FPGA**
Chip-Fläche	sehr klein	sehr groß
Anzahl Transistoren	sehr hoch	hoch bis sehr hoch
Taktrate	sehr hoch	mittel bis hoch
Entwicklungszeit	sehr lang	kurz
Kosten für Prototypen	sehr hoch	sehr niedrig
Stückkosten bei hoher Stückzahl	sehr niedrig	sehr hoch

Wie in der Tabelle 2.1 dargestellt, sind die Eigenschaften von kundenspezifischem SoC und FPGA über die Konfigurierbarkeit hinaus sehr verschieden. Während bei FPGAs die Konfigurierbarkeit durch den Kunden im Vordergrund steht, wird bei der Entwicklung von kundenspezifischen SoCs versucht, das Optimum für eine spezifische Anwendung und meist einen einzelnen Kunden zu erreichen. FPGAs bieten mittlere bis hohe Taktraten und verfügen über eine vergleichbare Anzahl an Transistoren, jedoch werden durch die Inkaufnahme großer Chipflächen deutlich schnellere Entwicklungszeiten und geringere Kosten in der Entwicklung ermöglicht.

Dagegen kann in der SoC-Entwicklung – trotz einer sehr hohen Anzahl an Transistoren von bis zu 10^9 – eine sehr kleine Chipfläche erreicht werden. Um dies zu ermöglichen, wird jeder kundenspezifische SoC bis runter zur Transistor-Ebene in seinem Platzbedarf optimiert. Die Transistor-Ebene steht für die unterste Abstraktionsebene des SoCs und wird in Abschnitt 2.6.2 näher beschrieben. Dies ist neben der hohen Komplexität und den hohen Anforderungen an die Performance (Leistungsfähigkeit) von SoCs ein Grund für die langen Entwicklungszeiten von im Schnitt fünf bis zehn Jahren. Die Entwicklung von kundenspezifischen SoCs lohnt sich daher erst ab einer Stückzahl von etwa 10^6, da sich hierbei die kleine Chipfläche positiv auf den Preis pro Stück auswirkt. Bei Stückzahlen über 10^6 sind die Stückkosten damit sehr gering, bei den FPGAs dementgegen sehr hoch [11] [12, pp. 659–665].

Neben der Chipgröße werden weitere nichtfunktionale Anforderungen an den SoC gestellt, welche seitens SoC-Entwicklung erfüllt werden müssen. Wichtige Beispiele hierbei sind:

- Die **Zuverlässigkeit** des SoCs ist besonders in der Anwendung in Kraftfahrzeugen entscheidend, da hier der Schutz von Menschenleben von der korrekten Funktionalität des SoCs abhängt.
- Die hohe Anzahl an SoCs in tragbaren Systemen und in Kraftfahrzeugen erfordert einen möglichst minimalen **Energiebedarf.**
- Das **Echtzeitverhalten** von SoCs im Automobilbereich ist in vielen Fällen – vergleichbar der Zuverlässigkeit – entscheidend für den Schutz von Menschenleben. Als Beispiel lässt sich hier das Antiblockiersystem (ABS) nennen. Die Reaktion des Systems in Echtzeit kann hierbei oftmals Schlimmeres verhindern.
- Die **Manipulationssicherheit** ist wie in allen Systemen, welche eine Interaktion mit der Anwenderin oder dem Anwender bieten, entscheidend. Ein Beispiel aus dem Bereich der Kraftfahrzeuge ist die Sicherheit gegen Manipulationen von Funkschlüsseln und Funkschlössern.

Die hier genannten Anforderungen sind lediglich ein exemplarischer Ausschnitt und variieren je nach Anwendungsfall in ihrer Priorisierung. Neben der Erfüllung der nichtfunktionalen Anforderungen müssen Automotive-SoC-Projekte in der Regel die Sicherheitsanforderungen nach der ASIL-Klassifikation erfüllen. Bei ASIL (Automotive Safety Integrity Level) handelt es sich um ein Risikoklassifizierungssystem, das durch die Norm ISO 26262 für die funktionale Sicherheit von Straßenfahrzeugen definiert ist [13]. Dabei wird eine Einstufung nach bestimmten

Parametern in die Stufen ASIL-A bis ASIL-D vorgenommen. Durch die steigende Interaktion der Teilsysteme und Funktionen im Automobil werden immer mehr elektronische Systeme als sicherheitsrelevant eingestuft und müssen somit nach dem Sicherheitsstandard ISO 26262 entwickelt werden. Die Einstufung stellt dabei auch die Automotive-SoC-Entwicklung vor weitere Herausforderungen [14, pp. 28–31].

Aufgrund des Fokus dieser Arbeit auf den Bereich der SoCs soll im nachfolgenden Abschnitt näher auf die typischen Teilsysteme und die Architektur eines SoCs eingegangen werden.

SoC-Architektur

Bei SoCs ist das gesamte System auf einem Chip realisiert. Daher beinhaltet der SoC in der Regel, zusätzlich zu den funktionalen Baugruppen, einen oder mehrere Mikroprozessoren. Die zu den Mikroprozessoren gehörenden Programme werden dazu auf einem Read-only-Memory (ROM) gespeichert. Abbildung 2.4 stellt eine typische Architektur eines SoCs als abstrahiertes Blockschaltbild dar.

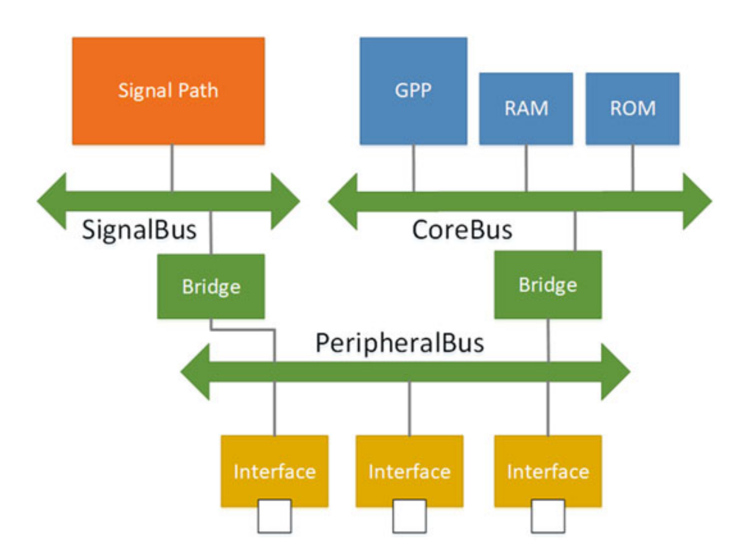


Abbildung 2.4 Abstrahierte Beispielarchitektur ASIC [15]

Die in der Abbildung 2.4 blau dargestellten Komponenten stellen das Mikrocontrollersubsystem des SoCs dar. In der Regel besteht dies aus dem General Purpose

Prozessor (GPP), einem RAM und einem ROM. Diese sind über den *CoreBus* miteinander verbunden. Die Komponenten zur Signalverarbeitung stellen ein weiteres Teilsystem dar und sind hier vereinfacht als *Signal Path* in Orange dargestellt. Auch hier werden die einzelnen Elemente später über den *SignalBus* verbunden. Die Schnittstellen nach außen werden in der Abbildung 2.4 abstrahiert als Interfaces in Gelb abgebildet. Sie stehen für die externen Schnittstellen des SoCs, wie zum Beispiel das Serial Peripheral Interface (SPI) oder das Controller Area Network (CAN). Die externen Schnittstellen werden über einen weiteren Bus, hier als *PeripheralBus* bezeichnet, verbunden. Die jeweiligen Teilsysteme werden über Busbrücken angeschlossen; diese übernehmen die Übersetzung der jeweils verwendeten Busprotokolle zueinander. Die Gesamtzahl der einzelnen Komponenten eines SoCs hängt dabei von den zu erfüllenden Aufgaben ab und ist in Art und Umfang sehr variabel [15].

2.3 SoC-Entwicklungs-Flow

Im folgenden Abschnitt wird der gesamte Zyklus von der Auftragserstellung bis hin zur Auslieferung des Produktes grundlegend behandelt. Dabei lassen sich übergeordnet in der Regel zwei Parteien definieren: der Auftraggeber, welcher den Entwicklungsprozess eines Produktes anstößt und seine Vorstellungen und Anforderungen an das Produkt im Lastenheft definiert, sowie der Auftragnehmer, welcher die Entwicklung eines Systems nach den gestellten Anforderungen zusichert. Der Auftragnehmer definiert meist in enger Abstimmung mit dem Auftraggeber das Pflichtenheft, in dem er beschreibt, wie er die Anforderungen des Kunden umsetzen wird. Dabei definiert er wiederum Anforderungen an das zu entwickelnde System, welche im Folgenden als Systemanforderungen bezeichnet werden. Da das Pflichtenheft in der Regel lediglich eine sehr abstrakte Beschreibung des zu entwickelten Systems enthält und damit nicht zur Realisierung des Systems ausreicht, kommen im Laufe der Entwicklung weitere detailliertere, spezifizierende Dokumente zum Einsatz. Diese Dokumente werden gemeinsam mit dem Lasten- und Pflichtenheft in dieser Arbeit als Spezifikationsdokumente bezeichnet. Die Definition der jeweiligen Spezifikationsdokumente und ihre Verwendung in den verschiedenen Phasen der SoC-Entwicklung wird im Nachfolgenden näher betrachtet. Angelehnt an das Modell nach Pahl/Beitz, welches vier Phasen für die Produktentwicklung definiert, sollen im Folgenden Phasen für die SoC-Entwicklung beschrieben werden [16] [14, pp. 34–35].

2.3.1 Planungsphase

Wie bereits erwähnt, beginnt die Entwicklung mit der Auftragserstellung durch den Auftraggeber, im Folgenden als Kunde bezeichnet. Dabei wird nach Lehrbuch das Lastenheft vom Kunden an den Auftragnehmer, im Folgenden als Entwickler bezeichnet, übergeben. Darin definiert der Kunde seine Vorstellungen über das zu entwickelnde System, definiert Anwendungsfälle, legt den technischen Kontext des Systems fest und definiert gegebenenfalls Soll- oder Grenzwerte. Die Analyse und adäquate Beschreibung dieser Informationen setzt jedoch ein gewisses Maß an Verständnis für die Technologie des zu entwickelnden Systems voraus. Daher findet bei der Entwicklung solch komplexer Systeme wie im Bereich der SoCs die Definition der Kundenanforderungen im Lastenheft oftmals bereits in Zusammenarbeit zwischen Kunde und Entwickler statt. Die genauen Inhalte des Lastenhefts sind in der DIN 69905 standardisiert. So sollte das Lastenheft neben den technischen Anforderungen auch qualitative und terminliche Anforderungen sowie auch Kostenanforderungen enthalten [14, p. 33] [16].

Auf Grundlage des Lastenheftes entwickelt der Auftragsnehmer bzw. Entwickler des Zielsystems das Pflichtenheft. Das Pflichtenheft dient zur Spezifikation des Systems, welches der Entwickler zu entwickeln gedenkt, um damit die Kundenanforderungen zu erfüllen. So können beispielsweise Performance-Eigenschaften, Toleranzbereiche und Grenzwerte zugesichert werden. Man spricht hierbei von Systemanforderungen. Nach allgemeiner Definition, wie in der DIN 69905, beschreibt das Pflichtenheft zudem, wie das System umgesetzt werden soll. Dies ist jedoch durch die hohe Komplexität in der SoC-Entwicklung zum Zeitpunkt der Pflichtenhefterstellung nur auf einer sehr abstrakten Ebene möglich. So werden die Funktionalitäten des Systems lediglich aus der Sicht des Kunden beschrieben. Zusätzlich zur Spezifikation des Systems dient das Pflichtenheft als Angebot an den Auftraggeber und bei Zustandekommen als Vertragsgrundlage für das Entwicklungsvorhaben. Dazu wird das Pflichtenheft zum Stand des Vertragsabschlusses in der Regel gemeinsam final geprüft und anschließend eingefroren [14, p. 33] [16].

2.3.2 Konzeptphase

Um Systemeigenschaften im Pflichtenheft zusichern zu können, erstellen die SoC-Architeken/-innentin parallel dazu ein erstes Konzept des Systems, nachfolgend als Systemkonzept bezeichnet. Das Systemkonzept besteht dabei meist aus einer formlosen Sammlung verschiedener Anforderungen, Tabellen, Notizen

und Zeichnungen, die den SoC auf System-Ebene beschreibt. Aufgrund der formlosen Art des Systemkonzepts kann das Systemkonzept im aktuell bestehenden Entwicklungs-Flow nicht als offizielles Spezifikationsdokument gesehen werden. Die Entwicklung des Systemkonzeptes geht weit über die Definition des Pflichtenhefts hinaus und ist entscheidend für eine erfolgreiche Systementwicklung. So wird bereits hier eine Vielzahl an Entwurfsentscheidungen, oftmals auf Grundlage von Annahmen über das spätere System, getroffen. Dabei werden einzelne Module definiert und in die entsprechenden Domänen unterteilt. Die gesamte Entwicklung findet hierbei jedoch rein konzeptuell statt; es wird in der Regel nicht implementiert. Der Begriff des Moduls wird in dieser Arbeit für eine funktionale Einheit im SoC verwendet, die Komponente dagegen entspricht einem bestimmten Hardwarebauteil, wie beispielsweise einem bestimmten Typ eines Mikrocontrollers, welcher im SoC verbaut werden soll.

Da bei der Entwicklung des Systemkonzeptes weitere Systemanforderungen entstehen, welche über das bereits eingefrorene Pflichtenheft hinausgehen, werden diese in dem darauf aufbauenden Spezifikationsdokument, hier als Konzeptspezifikation bezeichnet, dokumentiert. Die zusätzliche Spezifikation des Systemkonzepts ist entscheidend für die spätere Entwurfsphase und die darauffolgende Verifikation des Systems.

Erreicht das Systemkonzept und damit die Konzeptspezifikation einen gewissen Reifegrad, wird dieser Stand eingefroren; die Konzeptphase endet und die Entwurfsphase beginnt [16].

2.3.3 Übergang Konzeptphase – Entwurfsphase

Der Übergang zwischen Konzeptphase und Entwurfsphase führt zu einem Wechsel der Arbeitsweise. In der Konzeptphase findet die Entwicklung des SoCs rein konzeptuell auf Basis des Systemkonzeptes statt und wird meist von einem kleinen Team aus SoC-Architekt/-innen entwickelt; es wird nichts implementiert. Die Entwicklung endet in dieser Phase normalerweise auf Systemebene des SoCs.

In der Entwurfsphase steht die Implementierung des SoCs und seiner Module im Fokus. Dabei werden die maßgeblich in der Konzeptphase entwickelten Modulanforderungen an die Modulentwickler/-innen übergeben. Diese arbeiten meist alleine oder in kleinen Teams an einem Modul. Eine Projektleiterin oder ein Projektleiter, im Bereich der SoC-Entwicklung in der Regel als SoC-Architekt/-in bezeichnet, übernimmt die Koordination der einzelnen Entwicklerteams.

2.3.4 Entwurfsphase

Die Konzeptspezifikation beinhaltet die Anforderungen an die Module, welche an die Modulentwickler/-innen zu Beginn der Entwurfsphase übergeben werden. Diese beginnen mit der Entwicklung und Implementierung der jeweiligen Module. Dabei findet erneut eine Dekomposition des Moduls in Submodule, Gatter und schließlich in Transistoren statt. Während dieses Prozesses treffen die Modulentwickler/-innen Entwurfsentscheidungen, woraus neue Systemanforderungen entstehen. Die Dokumentation der Systemanforderungen erfolgt während der Entwurfsphase in der Entwurfsspezifikation, welche auf der Konzeptspezifikation aufbaut. Die Entwurfsspezifikation sollte daher zu jedem Zeitpunkt der Entwurfsphase ein möglichst genaues Abbild des implementierten Entwurfs darstellen.

In dieser Phase arbeiten bis zu 60 Entwickler/-innen parallel an der Implementierung der einzelnen Module. Da diese Module jedoch am Ende erfolgreich als Gesamtsystem interagieren müssen, ist es entscheidend, dass jeder Modulentwickler und jede Modulentwicklerin über den Systemkontext seines Moduls Bescheid weiß. Dazu dient in erster Linie das Systemkonzept. Da sich dieses jedoch in der Entwurfsphase ebenfalls weiterentwickelt, ist die Entwurfsspezifikation wichtigste Grundlage für einen Wissenstransfer zwischen den einzelnen Modulentwicklern/-innen. Die Koordination der einzelnen Entwicklungsarbeiten übernehmen die SoC-Architekten/-innen [16].

Da sich die Entwicklungszeit eines SoCs zwischen fünf und zehn Jahren bewegt und ein Großteil dieser Zeit der Entwurfsphase entspricht, ist es entscheidend, die Vollständigkeit und Richtigkeit der Entwurfsspezifikation neben dem Entwurf zu prüfen. Die dazu verwendeten Methoden werden in Abschnitt 2.3.5 beschrieben.

2.3.5 Simulation, Validation, Verifikation

Die Sicherstellung der Korrektheit und Vollständigkeit sowohl von Spezifikation als auch von Entwurf ist integraler Bestandteil der Entwicklung. Um dies zu ermöglichen, stehen unterschiedliche methodische Ansätze zur Verfügung, welche im Folgenden dargelegt werden.

Simulation
In der Simulation wird in der Regel durch das Simulieren definierter Szenarien die Abwesenheit von Fehlern überprüft. So lassen sich die wichtigsten Funktionalitäten eines Systems weitestgehend auf Fehlerfreiheit prüfen. Die Korrektheit des implementierten Entwurfs lässt sich hingegen durch die Simulation nicht garantieren, da lediglich eine begrenzte Zahl an Eingabefällen überprüft werden kann [17, pp. 295–296].

Validation
Die Validation ist in weiten Teilen Bestandteil des beschriebenen Reviews zwischen den Entwicklungsphasen. Sie dient zur Überprüfung der Spezifikation auf Vollständigkeit, Fehlerfreiheit, Konsistenz und Eindeutigkeit und prüft dabei, ob das Spezifikationsdokument der jeweiligen Phase den Kundenanforderungen entspricht. Bei der Validation können somit Spezifikationsfehler aufgedeckt werden. Dies ist entscheidend für die Verifikation [18].

Verifikation
Die Verifikation prüft, ob das Verhalten des Entwurfs der Spezifikation entspricht. Ist die Spezifikation nicht vollständig, können diese Bereiche nicht geprüft werden. Dadurch sind Fehler möglich, die meist erst in der Produktion oder beim Kunden entdeckt werden. Formale Verifikation meint die vollständige Verifikation des Systems, was sehr aufwendig und bei komplexen Systemen wie SoCs nur schwer umsetzbar ist [18].

Aufgrund der hohen Relevanz der Verifikation und Validierung für die vorliegende Arbeit und für die SoC-Entwicklung selbst werden die Verifikation und die Validierung in Abschnitt 2.6.8 ausführlich behandelt.

2.3.6 Abschluss SoC-Entwurf

Am Ende des Entwicklungs-Flows wird das System bzw. in diesem Fall der SoC erst an die Produktion und anschließend an den Kunden übergeben. Um das Produkt einsetzen zu können, benötigt der Kunde eine ausführliche Dokumentation, hier als technische Kundendokumentation bezeichnet. Diese stellt eine Teilmenge der Entwurfsspezifikation dar. Darin sollten dem Kunden alle Informationen, die später die Applikationsingenieure/-innen beispielsweise zur Interaktion mit dem System benötigen, zu Verfügung gestellt werden, ohne dabei jedoch zu viel Wissen über das System preiszugeben. Darüber hinaus enthält die technische Kundendokumentation oftmals Empfehlungen und Einschränkungen, wie

das System zu benutzen ist, um kein unerwartetes Verhalten zu provozieren. Damit kann die falsche Verwendung seitens des Kunden vermieden werden und der Entwickler bzw. Zulieferer sichert sich im Falle einer nicht vorhergesehenen Verwendung rechtlich ab.

2.3.7 Übersicht SoC-Entwicklungs-Flow

Bei den jeweiligen Spezifikationsdokumenten handelt es sich nicht um unabhängige Dokumente, sondern meist um eine Weiterentwicklung des Spezifikationsdokuments der vorangegangenen Phase, wie in Abbildung 2.5 veranschaulicht. Die namentliche Trennung der Spezifikationsdokumente dient in dieser Arbeit der Zuordnung der Spezifikationsdokumente zur jeweiligen Phase.

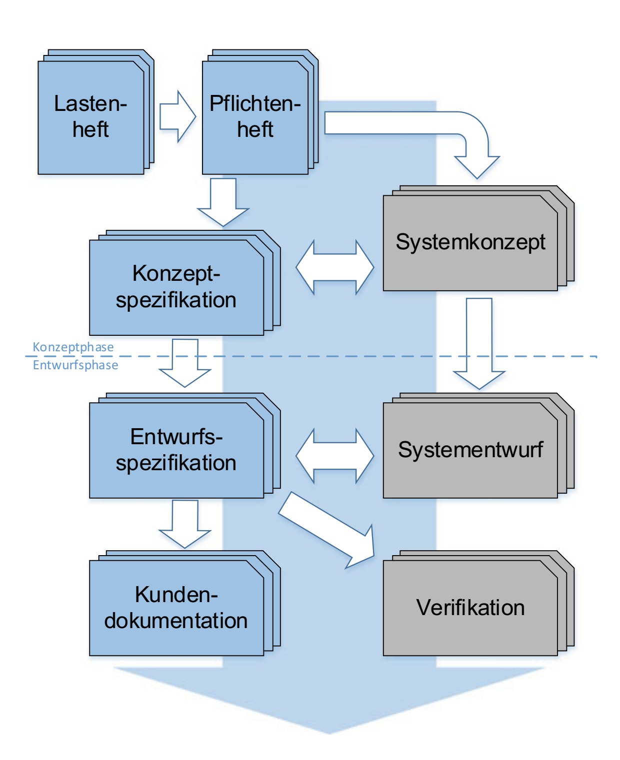


Abbildung 2.5 Spezifikationsdokumente im SoC-Entwicklungs-Flow

Die blau dargestellten Blöcke entsprechen den Spezifikationsdokumenten sowie der technischen Kundendokumentation. Die grauen Blöcke auf der rechten Seite des Entwicklungs-Flows den daraus entstehenden bzw. verlinkten Anteilen, welche jedoch nicht Teil der Spezifikation selbst sind. Die weißen Pfeile stehen für die weitestgehend manuellen Übergänge zwischen den Spezifikationsdokumenten selbst sowie zwischen jeweiligem Spezifikationsdokument und dem Systemkonzept bzw. Systementwurf. Die Validierung der Spezifikationsdokumente findet in der Regel bei den Übergängen zwischen diesen statt.

2.4 Fallstudie Anforderungsmanagement

Aus den in Abschnitt 2.1 gezeigten Trends und Besonderheiten der Automobilindustrie resultieren Herausforderungen für das Anforderungsmanagement. Diese Herausforderungen sollen anhand der Ergebnisse behandelt werden, die in einer Mehrfachstudie [19] erzielt wurden. Hierbei wurden sowohl Mitarbeiter/-innen eines großen OEM als auch Mitarbeiter/-innen von Zuliefererbetrieben befragt, um Defizite in Hinblick auf Kommunikation und Anforderungsmanagement zu ermitteln. Dazu wurden drei Applikationsingenieure/-innen, drei Systemingenieure/-innen sowie acht Ingenieure/-innen aus dem Bereich der Eingebetteten Systeme befragt.

Produkt- und Systemkontextverständnis

Um Systemanforderungen adäquat analysieren und beschreiben zu können, benötigt man ein umfassendes Wissen aller beteiligten Domänen sowie des Systemkontextes. Aufgrund der hohen Komplexität der Systeme und dem hohen Zeitdruck in der Entwicklung ist dieses Wissen oftmals nicht vorhanden. Hinzu kommt die Besonderheit der Automobilbranche, dass OEMs bei ihren Zulieferern Systeme beauftragen, über deren Systemkontext und Verwendung zu diesem Zeitpunkt ein hohes Maß an Unklarheit besteht. So ergab die in [19] durchgeführte Befragung die in Abbildung 2.6 veranschaulichte Zustimmungsrate in Bezug auf mangelnde Kenntnisse über das Produkt in frühen Phasen.

In den Abbildungen 2.6, 2.7, 2.8, 2.9 und 2.10 wird die Zustimmungsrate der jeweiligen Berufsgruppen prozentual gegenübergestellt. Dabei werden die Ingenieure/-innen aus dem Bereich Eingebettete Systeme links im Diagramm in Blau, die Systemingenieure/-innen im mittleren Graphen in Orange und die Applikationsingenieure/-innen im rechten Graphen in Grau dargestellt.

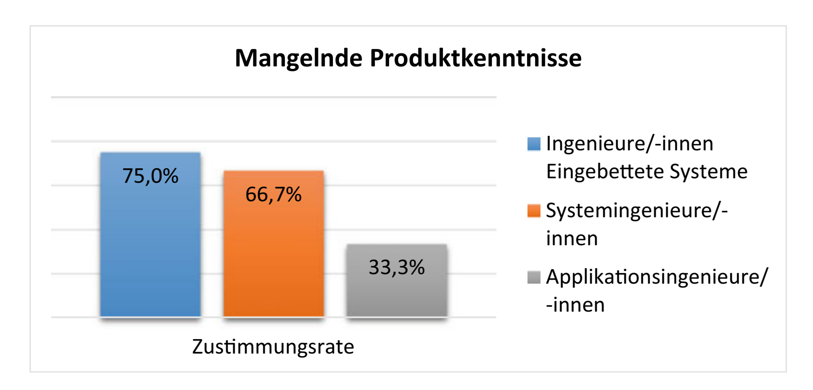


Abbildung 2.6 Zustimmungsrate: Mangelnde Produktkenntnis [19]

Bei der Befragung gaben 75 % der Ingenieure/-innen im Bereich der Eingebetteten Systeme, 66,75 % der Systemingenieure/-innen und 33,3 % der Applikationsingenieure/-innen an, Schwierigkeiten bei der Spezifizierung von Anforderungen in frühen Phasen zu sehen, welche aus einem fehlenden Wissen über das Produkt resultieren. Es lässt sich somit im Mittel ein hohes Maß an Zustimmung über alle drei Berufsgruppen hinweg erkennen. Anders verhält es sich bei der Frage über ein fehlendes Wissen des Systemkontextes.

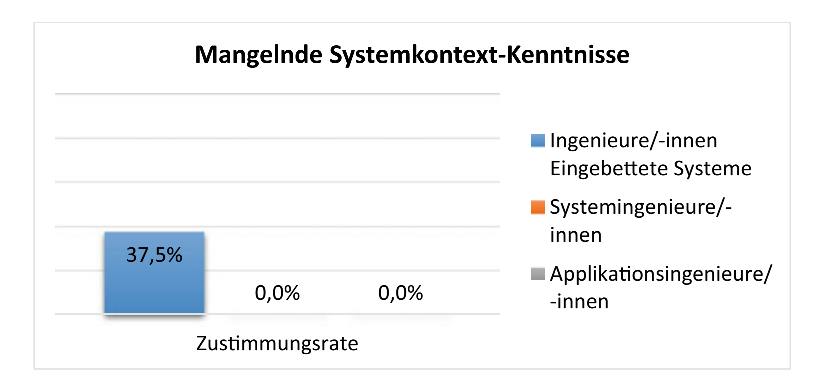


Abbildung 2.7 Zustimmungsrate: Mangelnde Systemkontextkenntnisse [19]

Hierbei sehen lediglich 37,5 % der befragten Ingenieure/-innen aus dem Bereich der Eingebetteten Systeme ein Fehlen von Wissen. Ein Grund für das fehlende Wissen über Systemkontext und Produkt seitens der Ingenieure/-innen aus dem Bereich der Eingebetteten Systeme liegt in der Pyramidenstruktur in der Automobilbranche und der resultierenden Marktmacht der OEMs gegenüber den Zulieferern. Über alle Hierarchieebenen hinweg besteht der Trend, Entwicklungsarbeiten an Zulieferer abzugeben, ohne dabei jedoch zu viel Informationen über das eigene Produkt preisgeben zu wollen. So schadet der Schutz der eigenen Innovationen dem Informationsfluss über die Zuliefererpyramide hinweg und somit oftmals der Qualität der Zusammenarbeit.

Umgekehrt fehlt dem Kunden als Anforderungssteller häufig Wissen über die Technologien der Zulieferer, um die Anforderungen eindeutig und adäquat beschreiben bzw. die Qualität der eigenen Anforderungen validieren zu können. Der lückenhafte Fluss der Informationen und das daraus resultierend fehlende Systemverständnis der beteiligten Geschäftsbereiche führt immer wieder zu Hindernissen bei der Kommunikation und der Zusammenarbeit.

Domänenübergreifende Zusammenarbeit

Nicht nur zwischen Kunden und Zulieferern kann es zu Fehlkommunikationen aufgrund unzureichender Spezifikation kommen. So arbeiten bei der Entwicklung heutiger Automotive SoCs in der Regel Entwickler/-innen aller Domänen eng zusammen. Um dies zu ermöglichen, ist ein hohes Maß an interdisziplinärem Verständnis gefragt. Abbildung 2.8 zeigt die in [19] erhaltene Zustimmungsrate zur Frage, ob die Ingenieure/-innen ein Fehlen dieses interdisziplinären Verständnisses sehen.

Dabei ist die Zustimmungsrate bei den Systemingenieuren/-innen besonders hoch. Aufgrund der hohen Komplexität der Systeme werden diese in Submodule und Domänen unterteilt und die Modulanforderungen an die jeweiligen Modulentwickler/-innen übergeben. Dabei fehlen oftmals ein interdisziplinäres Verständnis sowie Wissen über die Sprachen, Tools und Methoden der anderen Domänen und anderer Submodule, wodurch die Zusammenarbeit und die Kommunikation erschwert werden. Jedoch betreffen viele Modulanforderungen mehrere Domänen und Submodule gleichermaßen und so kann es durch ein fehlendes interdisziplinäres Verständnis zu Fehlinterpretationen der Anforderungen kommen.

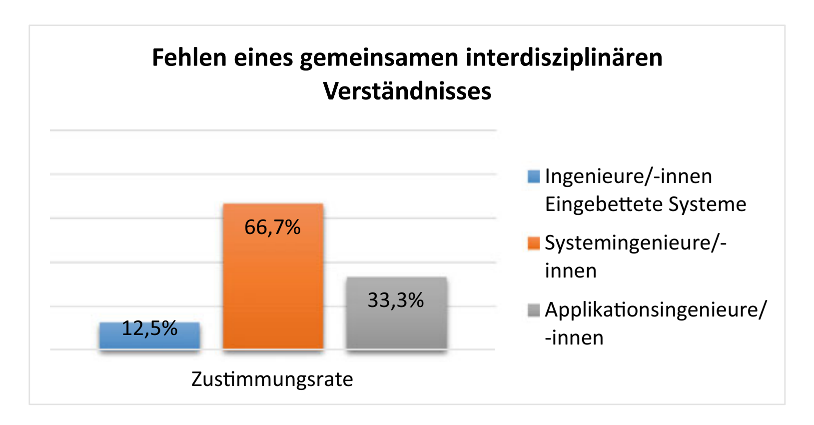


Abbildung 2.8 Zustimmungsrate: Fehlen interdisziplinäres Verständnis [19]

Dokumentation der Anforderungen

Als drittes Thema sollen die in [19] identifizierten Defizite in der Dokumentation und Verwaltung von Anforderungen diskutiert werden. Dabei veranschaulicht Abbildung 2.9 sehr allgemein die Zustimmungsrate zur Frage, ob in den jeweiligen Bereichen unzureichende Ressourcen für das Verstehen und die Pflege von Anforderungen bestehen.

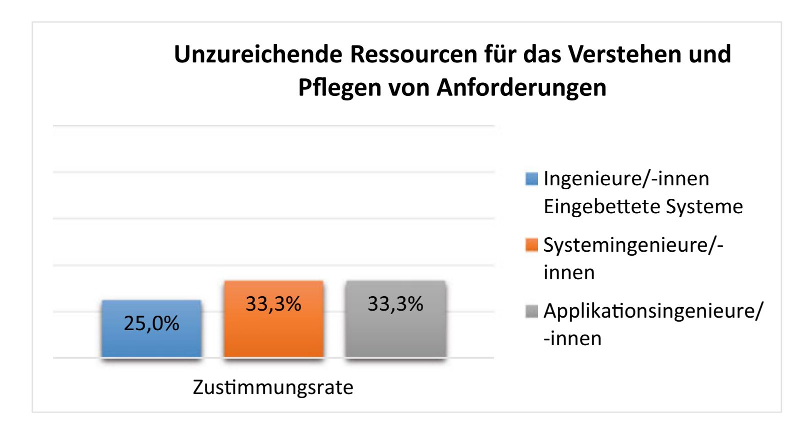


Abbildung 2.9 Zustimmungsrate: Unzureichende Ressourcen [19]

Dabei antworteten die Ingenieure/-innen des jeweiligen Bereichs einheitlich mit einer Zustimmungsrate von knapp unter einem Drittel mit „Ja“. Das Ergebnis der Befragung fällt somit geringer aus, als in Anbetracht der bestehenden Defizite im Anforderungsmanagement zu erwarten war. Es lässt sich somit der Schluss ziehen, dass die Defizite nicht etwa aufgrund von fehlenden Ressourcen entstehen, sondern auf Schwachstellen in den aktuell verwendeten Methoden und Werkzeugen zurückzuführen sind. Diese These wird durch die ermittelte Zustimmungsrate zur „Traceability“ (Nachverfolgbarkeit), wie in Abbildung 2.10 veranschaulicht, unterstützt. Bei dieser Befragung ging es um die Ermittlung, inwieweit Defizite in der Nachverfolgbarkeit der Anforderungen über Abstraktionsebenen hinweg bestehen.

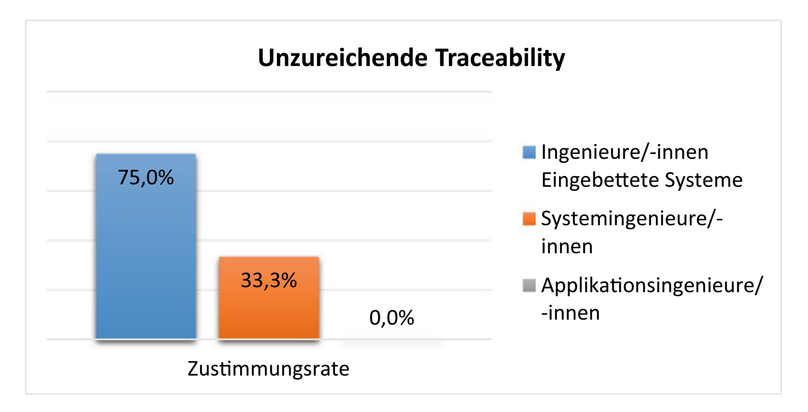


Abbildung 2.10 Zustimmungsrate: Unzureichende „Traceability“ (Nachverfolgbarkeit) [19]

Dabei stimmten 75,0 % der Ingenieure/-innen aus dem Bereich der Eingebetteten Systeme und 33,33 % der Systemingenieure/-innen einer unzureichenden „Traceability“ zu. Diese „Traceability“ steht für das In-Beziehung-Setzen der Anforderungen – von den Modul- bis hoch zu den Kundenanforderungen – und ist entscheidend, um den Modulentwicklern/-innen ein Verständnis über den Systemkontext und die übergeordneten Anforderungen zu vermitteln.

2.5 Modellbasierte Systementwicklung

Durch die steigende Komplexität von Hard- und Software heutiger Eingebetteter Systeme wird die Entwicklung solcher Produkte in interdisziplinären Teams zur Herausforderung. Entscheidende Faktoren für das Gelingen eines solchen Projektes sind hierbei die Fehlerfreiheit und Lesbarkeit der Spezifikation. Heutige Spezifikationen bestehen in der Regel aus Anforderungen, beschrieben mittels natürlicher Sprache sowie Schaubildern und Tabellen, welche direkt in die Spezifikation eingefügt werden. Diese Form der Spezifikation weist eine einfache Anwendbarkeit auf, da keine neuen Sprachen oder aufwendige Werkzeuge zur Beschreibung benötigt werden. Darüber hinaus jedoch weist die Spezifikation in natürlicher Sprache eine hohe Fehleranfälligkeit auf. Diese Fehleranfälligkeit resultiert aus der fehlenden Eindeutigkeit der natürlichen Sprache, den Defiziten bei der Überprüfbarkeit auf Vollständigkeit und Korrektheit sowie der unzureichenden oder nicht vorhandenen Formalisierung der Beschreibungsformen.

Ein Lösungsansatz, um diese Defizite in der Spezifikation zu minimieren oder zu beheben, ist die Anwendung der modellbasierten Systementwicklung. Hierbei erfolgen die Entwicklung und die Spezifikation von Systemen nicht mehr ausschließlich auf Basis von natürlicher Sprache, Tabellen und Schaubildern, sondern vorrangig mithilfe von Modellen. Unter einem Modell versteht man dabei eine abstrakte Repräsentation eines Systems oder eines Teils des Systems. Diese Definition soll jedoch im Kontext dieser Arbeit verfeinert werden. So bietet auch die Zeichnung eines Blockschaltbildes in einem Zeichentool eine abstrakte Repräsentation. Daher beinhaltet die Definition eines Modells für die hier vorliegende Arbeit die Verwendung einer standardisierten und maschinenlesbaren Modellierungssprache. Die Verwendung von grafischen Modellen ermöglicht durch die Visualisierung die schnellere Erfassung von Informationen und erlaubt zudem die Darstellung unterschiedlicher Sichtweisen auf das gleiche Modell. Eine Sicht auf einen Teil des Modells wird als Diagramm bezeichnet [20].

Zusätzlich zu den Vorteilen, welche aus der Visualisierung der Modelle entstehen, bieten modellbasierte Spezifikationen durch ihr maschinenlesbares Format die Möglichkeit der Simulation und der anschließenden Wiederverwendung der modellierten Informationen in Electronic Design Automation (EDA) Tools [21]. Für die Modellierung steht dabei eine große Anzahl verschiedener Modellierungssprachen zur Verfügung, wobei sich die hier vorliegende Arbeit aufgrund ihrer hohen Relevanz auf die objektorientierten Modellierungssprachen fokussiert. Nachfolgend werden daher die für diese Arbeit relevanten objektorientierten Modellierungssprachen behandelt.

2.5.1 UML

Die „Unified Modeling Language“ (UML) ist eine grafische objektorientierte Modellierungssprache und wurde ursprünglich für die Modellierung von Software entwickelt. Das Wort „Unified“ im Namen steht für die Verwendbarkeit der Modellierungssprache für die Beschreibung von Softwaresystemen jeglicher Geschäftsbereiche. Die Anpassbarkeit der Sprache wird erreicht, indem die Anwender/-innen eigene Typen, genannt *Stereotypen,* definieren und an die jeweilige Domäne anpassen können. Stereotypen dienen in der UML dazu, Modellelemente in ihren Eigenschaften und ihrer Semantik anzupassen.

Durch die Erweiterungen in der UML2 wuchsen die Einsatzmöglichkeiten der Modellierungssprache über die Domäne der Softwaresysteme hinaus. Die UML als weltweiter Industriestandard wird durch die Object Management Group (OMG) verwaltet und stetig weiterentwickelt [22, pp. 157–162] [23].

UML-Profil für MARTE
Für die Modellierung von Echtzeitsystemen und Eingebetteten Systemen kann die UML um ein MARTE-Profil erweitert werden. MARTE steht für „Modeling and Analysis of Real-Time and Embedded systems“. Profile sind neben den Stereotypen ein Erweiterungsmechanismus der UML. Ein Profil stellt in seiner Grundfunktion eine Menge aus Stereotypen dar. Durch Hinzufügen eines Profils können die enthaltenen Stereotypen auf die entsprechenden Modellelemente angewendet und dadurch angepasst werden. Darüber hinaus lassen sich beliebige Modellelemente und Diagramme in einem Profil hinterlegen. Das MARTE-Profil stellt eine gemeinsame Methode für die Modellierung der Hardware sowie auch der Software eines Eingebetteten Systems dar [24].

2.5.2 SysML

Die „Systems Modeling Language“ (SysML) entstand ehemals aus der in Abschnitt 2.5.1 beschriebenen UML und ist wie diese ein Industriestandard der OMG. Die SysML wurde entwickelt, um zusätzlich zur Software auch komplette Systeme zu modellieren, und kann damit bei allen System-Engineering-Anwendungen zum Einsatz kommen. Anfangs stellte die SysML lediglich eine Art Dialekt der UML dar, welche in vielen Fällen über vergleichbare Sprachmechanismen verfügt. Spätestens seit der Entwicklung der SysML 2.0 jedoch, wird sie meist als eigenständige Sprache betrachtet [22] [25]. So lässt sich in Abbildung 2.11 die Schnittmenge zwischen UML und SysML erkennen.

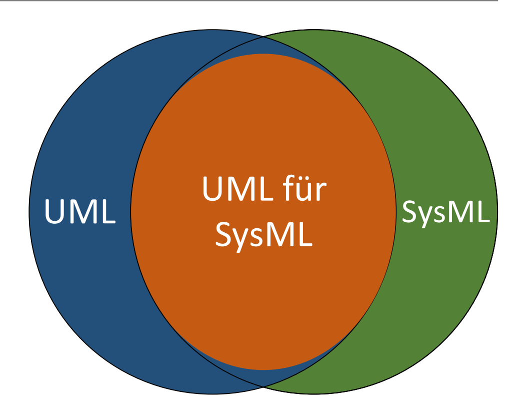


Abbildung 2.11
Schnittmenge UML/SysML
[25]

Zu erkennen ist der große Anteil der UML-Diagramme, welche identisch oder lediglich modifiziert in die SysML übernommen wurden. Oftmals wurde lediglich eine Verallgemeinerung der Bezeichnungen einzelner Diagramme und Elemente vorgenommen, um sich von dem Fokus der Softwareentwicklung zu lösen. Ein Beispiel ist die Umbenennung vom Klassen- ins Blockdiagramm und der Modellelemente Klassen in Blöcke. Diese verfügen im Grunde über die gleichen Eigenschaften; lediglich die Bezeichnung ermöglicht eine allgemeinere Verwendung. Die SysML stellt somit einen Werkzeugkasten aus Diagrammen und Modellelementen zur Verfügung. Darüber hinaus wurde jedoch keine Methodik standardisiert, wie die Diagramme zu verwenden sind. Diese fehlende Zuordnung von Diagrammen und Modellelementen zu einer spezifischen Bedeutung ermöglicht zum einen ein sehr breites Anwendungsfeld aufgrund der generischen Beschreibung, wird jedoch zum anderen dann zum Hindernis, wenn firmen-, team- und systemübergreifend gearbeitet werden soll. Ein mögliches Szenario wäre ein Modell einer Steuereinheit, welche in das übergeordnete Systemmodell des Fahrzeugs eingebunden werden soll. Hierbei müssen meist Modelle unterschiedlicher Entwicklungsteams zusammengesetzt werden, was sich ohne eine gemeinsame Modellierungsmethode nur mit erheblichem Aufwand realisieren lässt.

Im Nachfolgenden sollen die Diagramme der SysML und ihre übliche Verwendung behandelt werden. Die Abbildung 2.12 zeigt die enthaltenen Diagramme als Baumstruktur und veranschaulicht über die Farbgebung, welche Diagramme aus der UML2 übernommen, welche modifiziert und welche neu entwickelt wurden.

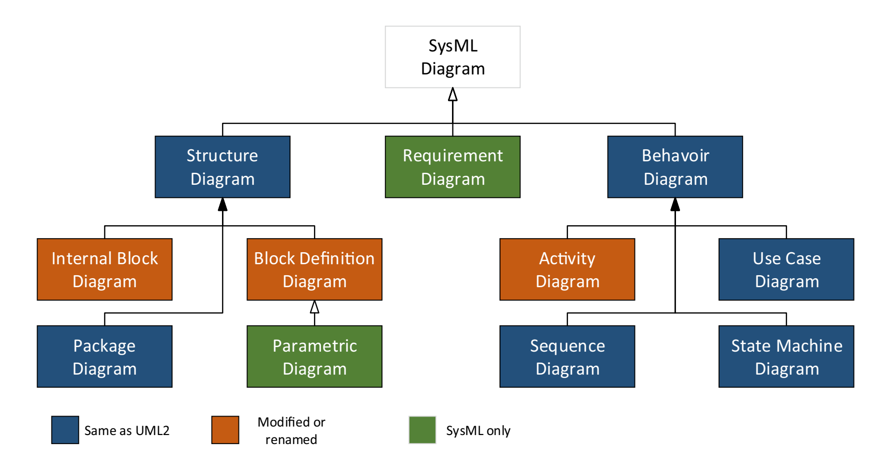


Abbildung 2.12 SysML-Diagramme [25]

Die Diagramme lassen sich in drei Oberkategorien unterteilen: die Verhaltensdiagramme, die Strukturdiagramme und das Anforderungsdiagramm. Die grün hinterlegten Anteile stehen für die Diagramme, welche neu in der SysML hinzugefügt wurden. Die orangen Blöcke stehen für Diagramme, welche verglichen mit der UML verändert wurden und die blauen Diagramme sind identisch wie in der UML2.

Strukturdiagramme

- Das **Block Definition Diagram** (BDD) wird in der Regel verwendet, um eine Dekomposition eines Systems bzw. einer Funktion abzubilden. Das heißt, es wird zum Beispiel dargestellt, aus welchen Komponenten und Unterkomponenten das System Auto, wie in Abbildung 2.13 dargestellt, besteht [22].

- Das **Internal Block Diagram** (IBD) ermöglicht die Beschreibung einer Architektur jeder Art mit den enthaltenen Komponenten, Schnittstellen und Verbindungen [22] [25]. Abbildung 2.14 zeigt die Komponenten des Autos aus Abbildung 2.13 als IBD.

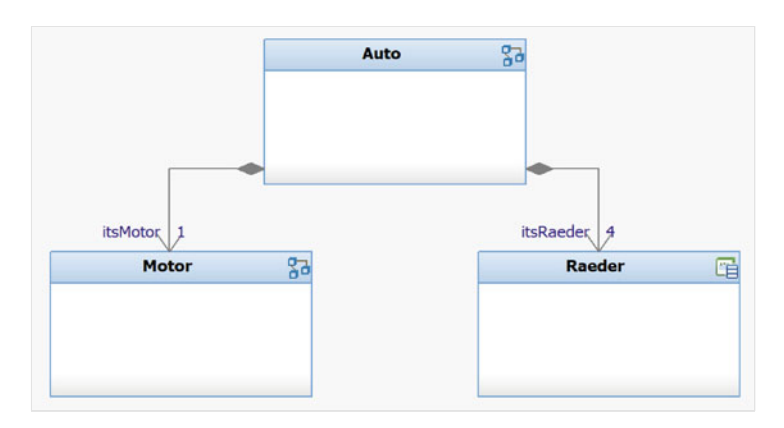


Abbildung 2.13 Beispiel eines Block Definition Diagram (BDD)

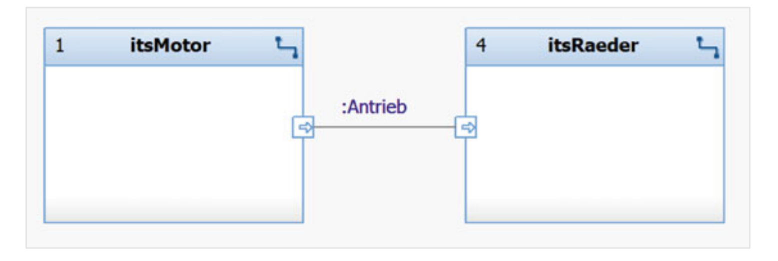


Abbildung 2.14 Beispiel eines Internal Block Diagram (IBD)

- Das **Package Diagram** wird anders als die übrigen Diagramme nicht zur Beschreibung des Systems eingesetzt, sondern kommt bei der Strukturierung des Modells zum Einsatz. So kann im Modell eine Art Ordnerstruktur mittels Packages aufgebaut werden, wodurch zusätzlich die Navigation im Diagramm erleichtert werden kann. Neben der Strukturierung des Modells lassen sich Packages im **Package Diagram** in Beziehung setzen. Somit dient dieses Diagramm dazu, die Bedeutung des jeweiligen Package zu erläutern und den Anwendern/-innen eine Orientierung in komplexen Modellen zu erleichtern [22] [25]. Abbildung 2.15 dient als Beispiel eines **Package Diagram**.

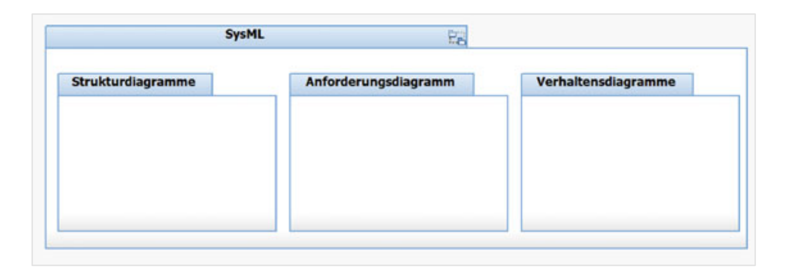


Abbildung 2.15 Beispiel eines Package Diagram

- Das **Parametric Diagram** stellt einen Diagrammtyp dar, welcher im Rahmen der SysML-Spezifikation definiert wurde. Es dient der Darstellung von mathematischen Beschreibungen und wertbedingten Einschränkungen, je nach Systemeigenschaften. Es wird oft zur Darstellung von analysebezogenen Aspekten und Rahmenbedingungen verwendet [22] [25]. Abbildung 2.16 zeigt als Beispiel eines Parametric Diagram den **ConstraintBlock** (Masse-Beziehung).



Abbildung 2.16 Beispiel eines Parametric Diagram

Anforderungsdiagramm

- Das **Requirement Diagram** ist eine Besonderheit der SysML. Es soll eine Möglichkeit bieten, die bei der Systementwicklung sehr wichtigen Anforderungen in Beziehung zu setzen. Dabei kann es dazu dienen, die Beziehungen zwischen Anforderungen darzustellen oder die Anforderungen mit Modellelementen anderer Diagramme in Beziehung zu setzen, wie in Abbildung 2.17 veranschaulicht [22] [25].

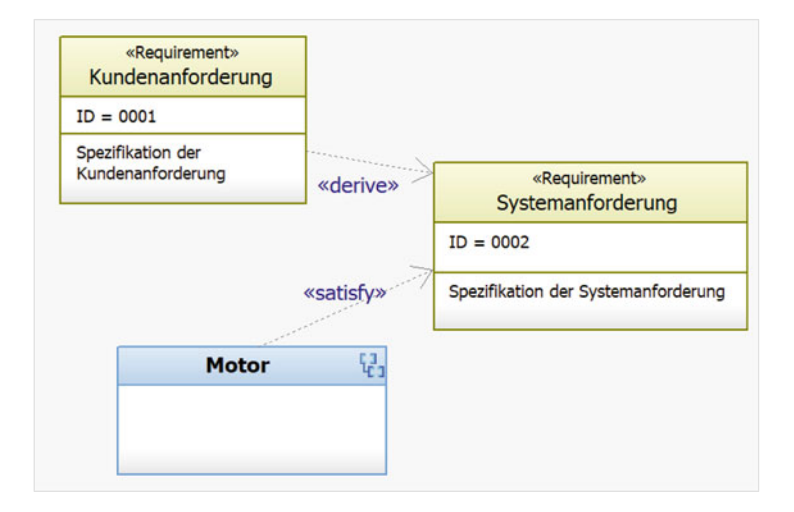


Abbildung 2.17 Beispiel eines Requirement Diagram

Verhaltensdiagramme

- Das **Activity Diagram** dient zur Beschreibung des Systemverhaltens, wobei der Fokus dieses Diagramms auf der Modellierung von Daten- und Kontrollfluss liegt. Es wird in der Systementwicklung meist für die Abbildung des Verhaltens einer einzelnen Komponente oder Funktion benutzt [22] [25]. Abbildung 2.18 zeigt ein Beispiel für ein Activity Diagram.

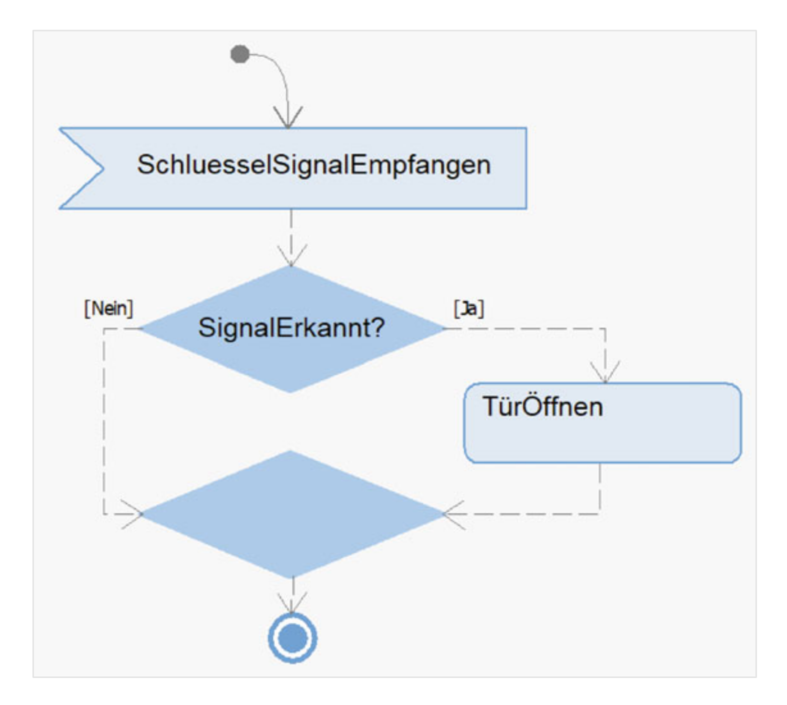


Abbildung 2.18 Beispiel eines Activity Diagram

- Das **State Machine Diagram** dient, wie in Abbildung 2.19 dargestellt, zur Modellierung von Zuständen und deren Übergängen eines Systems. Das **State Machine Diagram** kann zur Modellierung jedes beliebigen Systems genutzt werden, da alle Systeme über Zustände und Zustandsübergänge verfügen [22] [25].

- Das **Sequence Diagram** bildet ebenfalls das Verhalten eines Systems als Ablauf ab, konzentriert sich hierbei jedoch sehr stark auf die Kommunikation und den Datenfluss zwischen einzelnen Komponenten im System, wie in Abbildung 2.20 veranschaulicht [22] [25].

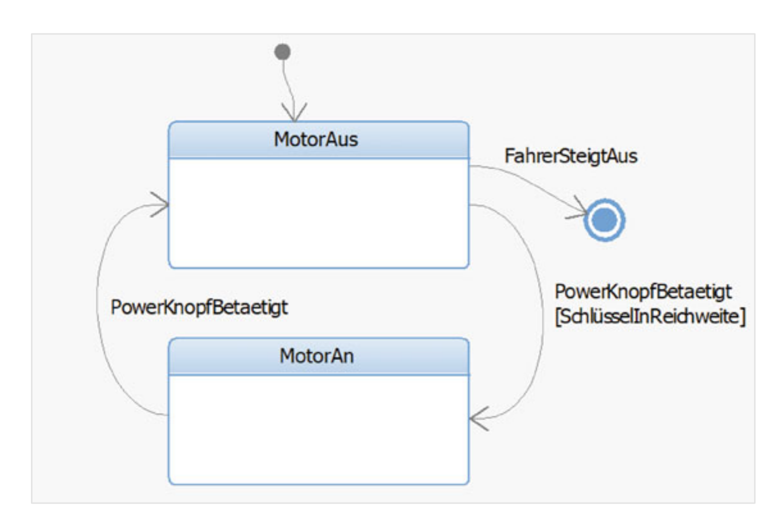


Abbildung 2.19 Beispiel eines State Machine Diagram

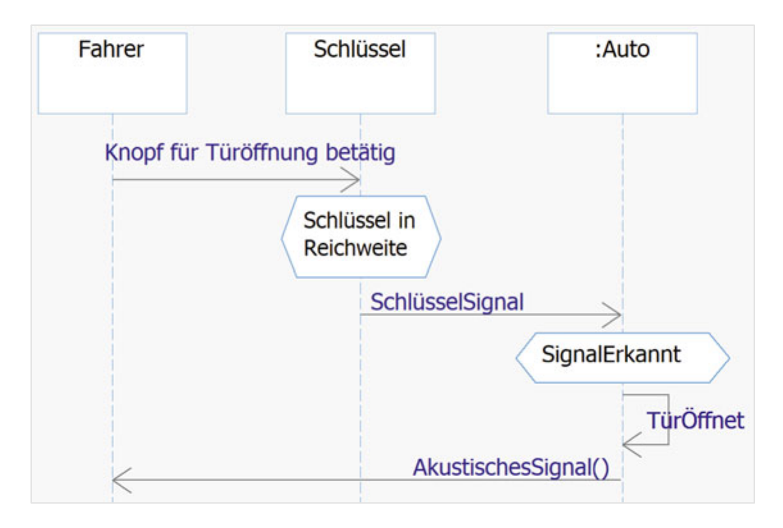


Abbildung 2.20 Beispiel eines Sequence Diagram

- Das **Use Case Diagram** ist in der Gruppe der Verhaltensbeschreibung für sich zu sehen. So geht es weniger darum, ein Verhalten zu modellieren, als vielmehr darum, die Funktionen des Systems aus Sicht der Anwender/-innen zu betrachten und die Nutzer des jeweiligen Anwendungsfalls zu definieren [22] [25]. Abbildung 2.21 zeigt ein stark abstrahiertes Beispiel eines **Use Case Diagram**.

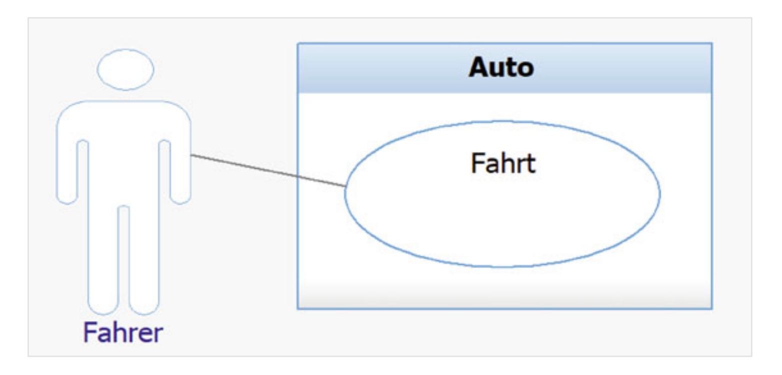


Abbildung 2.21 Beispiel eines Use Case Diagram

2.5.3 DSML

Zusätzlich zu bestehenden Modellierungssprachen lassen sich sogenannte „Domain-Specific Modeling Languages" (DSMLs) definieren. Dabei wird eine Modellierungssprache speziell für eine Domäne entwickelt und definiert, um sie ideal an die Anforderungen der Domäne anzupassen. So bieten DSMLs bei der Modellierung eine hohe Anwenderfreundlichkeit, da die Modellelemente bereits mit Fachbegriffen versehen sind und die Anwender/-innen diese daher nicht erst rekonstruieren müssen. Darüber hinaus sollte aufgrund einer sorgfältigen Analyse der Anforderungen bei der Entwicklung der DSML diese keine Modellelemente oder Diagramme enthalten, welche nicht von den Anwendern/-innen benötigt werden. Für eine bessere Verständlichkeit der DSML-Modelle sorgt die grafische Anpassung der Modelle an die Domäne [26].

Neben den Vorteilen einer DSML lassen sich besonders drei Nachteile nennen:

- Die Wirtschaftlichkeit bei der Analyse und Entwicklung einer DSML
- Die benötigte Wartung aufgrund sich verändernder Anforderungen an die DSML
- Eine in der Regel fehlende Tool-Unterstützung bei der Anwendung der Methode [26]

2.5.4 Formalisierung

Die Literatur bietet verschiedene Definitionen für den Vorgang der Formalisierung einer Modellierungssprache. In dieser Arbeit wird die Definition aus dem ISO/IEC/IEEE 31320–2-Standard zugrunde gelegt. Die Formalisierung wird hierbei als Definition einer genauen Semantik und damit einer Zuweisung von Modellelementen bzw. Diagrammen zu einer formalen Bedeutung beschrieben. Sie ermöglicht damit die genau definierte Interpretation aller Bestandteile eines Modells. Darüber hinaus beinhaltet die Formalisierung eine Einschränkung der zu verwendenden Modellelemente und Diagramme einer Modellierungssprache. Dabei dienen Metamodelle zur Beschreibung der zulässigen Diagramme, zur Definition dessen, was ein jeweiliges Diagramm enthalten darf, sowie zur Definition der Bedeutung der jeweiligen Modellelemente. Für das Modellieren der Metamodelle wird dabei meist die zu formalisierende Sprache verwendet, wodurch auch hier die Gefahr von Interpretationsfehlern entsteht. Um dies zu verhindern, eignet sich die Verwendung einer Prädikatenlogik als formale Sprache zur eindeutigen Beschreibung der Zuordnung zwischen Modellelement und formaler Bedeutung. Die Beschreibung mittels Prädikatenlogik erschwert hingegen jedoch den Wissenstransfer der Formalisierungsregeln zu den Anwendern/-innen, wodurch sich eine Kombination der beiden Ansätze empfiehlt: die eindeutige Beschreibung mittels Prädikatenlogik und die zusätzliche Veranschaulichung der Zusammenhänge als Metamodell für die Anwender/-innen.

2.6 Entwicklungsmethodik

Als Ergänzung zu der in Abschnitt 2.3 beschriebenen Entwurfsphase werden nachfolgend Grundlagen für den Entwurf von Automotive SoCs behandelt. Dabei werden zu Beginn die Entwurfsdomänen und Abstraktionsebenen der SoC-Entwicklung definiert sowie die Grundlagen des Top-down-Ansatzes und der

Entwicklung von Virtuellen Prototypen. Anschließend wird auf die Grundlagen des Hardware-, des Software- sowie des Hardware/Software-Co-Entwurfs eingegangen. Zum Schluss beschäftigt sich der Abschnitt mit der Verifikation und Validierung in der SoC-Entwicklung.

2.6.1 Domänen

SoCs sind heterogene Systeme, das heißt, sie setzen sich aus Teilsystemen zusammen, deren Datenverarbeitung auf grundsätzlich unterschiedlichen Signalarten beruhen. SoCs lassen sich in die Domänen Analog- und Digitaltechnik unterteilen. In der Analogtechnik basiert die Datenübertragung und -verarbeitung auf Basis von Spannungen und Strömen. In der Digitaltechnik erfolgt die Datenverarbeitung rein digital. Die analogen Anteile des SoCs werden vollständig durch Hardware realisiert. Dementgegen lassen sich die digitalen Anteile des SoCs in einen digitalen, festverdrahteten Hardware-Anteil und einen digitalen, programmierbaren Software-Anteil unterteilen. Aufgrund der damit einhergehenden unterschiedlichen Anforderungen werden für die Entwicklung unterschiedliche Entwurfsmethoden benötigt [14, p. 57]. Es ergeben sich drei Entwurfsdomänen:

- Analoge Hardware
- Digitale Hardware
- Software

Viele Funktionen des SoCs lassen sich sowohl in Hardware als auch in Software realisieren. Somit muss im Laufe des Entwicklungsprozesses diese Partitionierung durch die Entwickler/-innen erfolgen. Dies geschieht in der Regel bereits in der in Abschnitt 2.3.2 beschriebenen Konzeptphase. Dabei unterscheiden sich Hardware und Software maßgeblich in den folgenden Kriterien:

- Die Realisierung als spezifische Hardware ermöglicht eine schnelle und zuverlässige Ausführung komplexer Funktionen und Algorithmen. Die Entwicklung im Bereich der SoCs ist jedoch meist aufwendig und Anpassungen zu einem späteren Zeitpunkt sind nur schwer möglich.
- Die Realisierung in Software dagegen ermöglicht eine höhere Flexibilität und bietet daher in der Regel eine kostengünstige und platzsparende Alternative für die Realisierung von Funktionen, für die die Ausführungsgeschwindigkeit eines Standardprozessors ausreicht [14, p. 57].

2.6.2 Abstraktionsebenen

Aufgrund der hohen Komplexität heutiger Hardwaresysteme, besonders im Bereich der SoC-Entwicklung, findet der Entwurf auf mehreren Abstraktionsebenen statt. Dabei beginnen Entwickler/-innen in der Regel auf der höchsten Abstraktionsebene, d. h. auf Grundlage eines stark abstrahierten Systems bzw. SoCs, und verfeinern die Struktur und Verhaltensbeschreibung in mehreren Iterationsstufen bis zur Transistor-Ebene, welche den Abschluss des Entwicklungsprozesses darstellt. Das Y-Diagramm von Gajski und Kuhn definiert dazu fünf Abstraktionsebenen bzw. Entwicklungsstufen [27]. In Anlehnung an das Y-Diagramm werden nachfolgend Abstraktionsebenen aus Sicht der Struktur, bezogen auf die Automotive-SoC-Entwicklung, für diese Arbeit definiert:

- Die **System-Ebene** zeigt die Dekomposition des SoCs in einzelne Module und Submodule sowie deren Verbindungsstruktur.
- Auf der **Algorithmen-Ebene** werden Einheiten definiert, welche Algorithmen ausführen, wie zum Beispiel der Mikroprozessor des SoCs.
- Auf der **Register-Transfer-Ebene** (RT-Ebene) werden Register und RT-Einheiten wie Addierer, Multiplexer und arithmetisch-logische Einheiten (ALU) bestimmt.
- Die **Gatter-Ebene** beinhaltet die Beschreibung der Logik-Gatter.
- Die nachfolgend als **Transistor-Ebene** bezeichnete Ebene, stellt die tiefste Abstraktionsebene dar. Hier setzen sich die Gatter aus den einzelnen Transistoren zusammen. Eine Optimierung des Platzbedarfs findet auf Transistor-Ebene statt [14, pp. 48–52].

Die in Abschnitt 2.3.2 beschriebene Konzeptentwicklung findet sowohl für die Funktions- als auch für die Struktursicht in erster Linie in der System-Ebene statt. Jedoch werden als Teil des Systemkonzepts in der Regel zudem Teile der Algorithmen- und der RT-Ebene beschrieben, wie beispielsweise die Register des SoCs.

2.6.3 Top-down-Ansatz

Werden die im vorigen Abschnitt beschriebenen Abstraktionsebenen strikt sequenziell bearbeitet, spricht man von einer Entwicklung nach dem Top-down-Ansatz. Ganz allgemein sieht der Top-down-Ansatz eine Bearbeitung eines Problems anfangs auf einer stark abstrahierten Ebene vor. Anschließend soll in

mehreren iterativen Schritten das Problem dekomponiert werden, um eine Menge kleinerer und leichter lösbarer Teilprobleme zu erhalten [14, p. 51].

Überträgt man diesen Ansatz auf die SoC-Entwicklung, stehen zu Beginn statt eines Problems die Kundenanforderungen bzw. eine geforderte Funktionalität des SoCs. Diese sind in der Regel sehr abstrakt beschrieben und müssen zu Beginn durch die Entwickler/-innen analysiert und dekomponiert werden. Um Kundenanforderungen zu analysieren, empfiehlt sich zu Beginn eine Anwendungsfallanalyse. Da Nutzer des Systems oftmals andere Systeme oder Komponenten sind, muss parallel eine Analyse des technischen Systemkontextes durchgeführt werden. Das bedeutet, es findet eine technische Betrachtung der umliegenden Komponenten und Systeme statt, mit denen der SoC später interagiert. Die Analysen dienen zudem dazu, ein einheitliches Verständnis über die Nutzung und den Kontext des geforderten Systems zu erlangen. Aus den hierbei gewonnenen Anwendungsfällen lassen sich Funktionen ableiten, welche das System den Nutzern zur Verfügung stellen muss. Um diese Funktionen zu realisieren, bedarf es wiederum einer Menge an Teilfunktionen, die durch eine Dekomposition analysiert werden können. Dieser Schritt soll jedoch losgelöst von der Auswahl möglicher Hardware-Komponenten des SoCs oder anderer Entwurfsentscheidungen erfolgen. Durch dieses Vorgehen wird zum einen die Wahrscheinlichkeit reduziert, Funktionen nicht zu berücksichtigen, zum anderen werden keine Funktionen realisiert, die nicht vom Kunden gefordert wurden.

Hat die Dekomposition der Funktionen eine gewisse Tiefe erreicht, kann im nächsten Schritt ein Systemkonzept entwickelt werden. Dabei werden Funktionen in Module zusammengefasst und erste Entwurfsentscheidungen über die Architektur des SoCs getroffen. Der SoC wird in dieser Phase in der Regel lediglich abstrahiert auf System-Ebene entwickelt. Während der Entwicklung des Systemkonzeptes entstehen Modulanforderungen, welche in mehreren Modulspezifikationen zusammengefasst werden. Diese Modulspezifikationen werden zu Beginn der Entwurfsphase an die jeweiligen Modulentwickler/-innen übergeben, welcher mit der Implementierung der Hardware bzw. Software beginnen.

2.6.4 Virtuelle Prototypen

Ein Werkzeug des Top-down-Ansatzes ist die Entwicklung von Hardware-Prototypen noch vor Fertigstellung der eigentlichen Hardware. Da hierbei die Hardware – meist auf System-Ebene – lediglich abstrahiert als Prototyp realisiert wird, kann so die Simulation von Hard- und Software zu einem früheren

Zeitpunkt der Entwicklung erfolgen. Es wird dabei lediglich eine Teilfunktionalität der Hardware im Prototyp nachgebildet, welche für die Interaktion mit der Software benötigt wird.

Eine Möglichkeit der Realisierung von Hardware-Prototypen ist die Implementierung mittels FPGA. FPGA-Prototypen bieten eine sehr zuverlässige Möglichkeit der Simulation. Dementgegen steht die aufwendige Implementierung des FPGA als Prototyp, welche zudem meist erst in einer späten Phase der Hardware-Entwicklung erfolgt. Einen weit flexibleren Ansatz bietet die Implementierung des Prototyps mittels SystemC als sogenannter Virtueller Prototyp (VP) [15]. Bei SystemC handelt es sich um ein Hardware-Beschreibungsframework und eine Bibliothek der Sprache C++, welche die Implementierung von Virtuellen Prototypen ermöglichen. SystemC bietet bei der Simulation von Hardware die Möglichkeit einer Abstrahierung im Vergleich zu anderen Hardware-Beschreibungssprachen und damit eine bessere Performance bei der Simulation. Die Möglichkeit der Abstrahierung ist ein entscheidender Vorteil für die Entwicklung von Virtuellen Prototypen, da so bereits vor Fertigstellung eines vollständigen und finalen Entwurfs der Hardware erste Konzepte der Hardware und Software gemeinsam getestet werden können. Darüber hinaus bedeutet die Abstrahierung zwar in der Regel einen Verlust der Möglichkeit einer Taktzyklen-genauen Simulation, wobei eine solche Taktzyklen-genaue Ausführung der Simulation im Zusammenhang mit Virtuellen Prototypen meist nicht gefordert beziehungsweise nicht benötigt wird [28] [29]. Die Implementierung des Virtuellen Prototyps kann bereits parallel zur Spezifizierung des SoCs begonnen werden, wie in Abbildung 2.22 veranschaulicht.

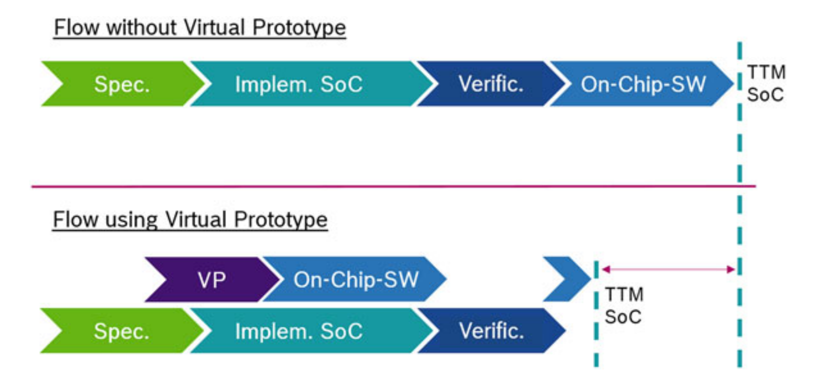


Abbildung 2.22 VP-Entwicklungs-Flow

Die Abbildung 2.22 zeigt abstrahiert den Entwicklungs-Flow eines SoCs. Im ersten Flow wurde kein Virtueller Prototyp eingesetzt und so erfolgt die Entwicklung der Hardware und Software vollständig sequenziell. Im zweiten Flow wird bereits parallel zur Erstellung der Spezifikation mit der Entwicklung des Virtuellen Prototyps begonnen und somit eine parallele Entwicklung von Software und Hardware ermöglicht. Zum Abschluss der Entwicklung erfolgt die Systemintegration und damit die Zusammenführung von Hardware und Software, in der Abbildung 2.22 als kurzer hellblauer Pfeil am Ende des Flows dargestellt. Durch den Einsatz der Virtuellen Prototypen kann im Idealfall eine Reduktion der Time-to-Market (TTM, Zeit bis zur Markteinführung), also der Zeit, die das Produkt von Auftragserteilung bis Marktfreigabe benötigt, erreicht werden.

Zusätzlich ermöglicht dieser Ansatz eine Validierung des Hardware/Software-Interfaces und somit die Entwicklung von Hardware und Software in ständigem Austausch über Änderungen am Systemkonzept. Dies ermöglicht, dass Fehler, welche bei einer linearen Entwicklung sonst erst in einer deutlich späteren Phase gefunden werden, früher entdeckt und somit leichter behoben werden können. Hierdurch können somit bei der Entwicklung zusätzlich Aufwand und Kosten reduziert werden.

2.6.5 Softwareentwicklungsmethodik

Aufgrund der steigenden Anforderungen und der daraus resultierenden Komplexität der SoCs gewinnt die On-Chip-Software immer stärker an Bedeutung. So bietet die Realisierung des SoC-Verhaltens als Software eine einfachere und damit kostengünstigere Alternative für die Entwicklung und ermöglicht, verglichen mit der Realisierung mittels spezifischer Hardware, zudem nachträgliche Korrekturen und Erweiterungen bei minimalem Aufwand. Lediglich die Ausführungszeit der Standardprozessoren, welche die Software ausführen, ist in der Regel langsamer, wodurch besonders im Bereich der Automotive SoCs weiterhin Teile als spezifische Hardware realisiert werden müssen.

Die Realisierung der Software für Eingebettete Systeme erfolgt in einer maschinenlesbaren höheren Programmiersprache wie beispielsweise C. Anschließend übersetzt der Compiler die Software in Maschinencode bzw. Assembler-Code. Da es sich bei SoCs oftmals um Echtzeitsysteme handelt, welche strengen Beschränkungen unterliegen, ergeben sich auch für die Softwareentwicklung in diesem Bereich zusätzliche Anforderungen. So führt die Beschränkung des Platzbedarfs des SoCs zu einer Begrenzung des zulässigen Speichers und die

Forderung nach einem minimalen Energiebedarf erfordert eine aufwendige Optimierung der Software-Performance. Hinzu kommt, dass die Entwicklung der Software stark von der Hardware abhängt und so die Softwareentwicklung in vielen Fällen erst in einer späten Phase der Entwicklung starten kann.

Für die Entwicklung komplexer Software haben sich heute meist sogenannte „agile" Entwicklungsprozesse durchgesetzt. Dabei wird die Software nicht zu Beginn vollständig geplant und erst anschließend realisiert. Vielmehr wird die Software in mehreren iterativen Schritten entwickelt und implementiert. Hierdurch können erste Softwarekonzepte in Verbindung mit der Hardware überprüft und bei Bedarf sehr flexibel angepasst werden.

Seit einigen Jahren geht der Trend in der Entwicklung von Software für Eingebettete Systeme in Richtung der Softwaregenerierung. Dadurch kann die aufwendige und fehlerbehaftete manuelle Programmierung der Software zumindest teilweise entfallen [14, pp. 38–44].

2.6.6 Hardwareentwicklungsmethodik

Die Hardwareentwicklungsmethoden erfuhren in den vergangenen Jahrzehnten mehrere evolutionäre Entwicklungsschritte. Heute werden komplexe Systeme in der Regel nach der „Spezifizieren, Explorieren und Verfeinern (SER)"-Entwicklungsmethode entwickelt. Dabei werden die Anforderungen an das System zu Beginn umfangreich spezifiziert und auf Grundlage des Spezifikationsdokumentes wird in natürlicher Sprache ein Modell auf System-Ebene erstellt. Hierdurch lässt sich bei möglichst vollständiger Modellierung der zu Beginn textuell beschriebenen Funktionen eine simulierbare Spezifikation erreichen. Die erreichte Simulierbarkeit der Spezifikation ermöglicht die Überprüfung und Validierung erster Systemkonzepte bereits in einer sehr frühen Phase der Entwicklung. Dabei lassen sich verschiedene Konzepte mit verschiedenen Systemeigenschaften als Modell erstellen und durch die Simulation vergleichen. Man spricht von einer Exploration. Darauf aufbauend können Entwurfsentscheidungen getroffen werden und so das Konzept bzw. der Entwurf iterativ verfeinert werden [14, pp. 55–57].

Plattformbasierter Entwurf

Die Evolution der Entwurfsmethodik geht einher mit der Evolution der SoC-Hardware. Heutige SoCs setzen sich aus Millionen von Gattern und damit aus mehreren Milliarden Transistoren zusammen. Zur Beherrschung dieser Umfänge in der Entwicklung wird heute, wenn möglich, auf einen plattformbasierten

Entwurf zurückgegriffen. Plattformbasiert steht hierbei für die Zusammensetzung eines Systems aus einer Reihe vorgefertigter und zueinander kompatibler Komponenten. Dabei kommen neben Standardkomponenten sogenannte „Intellectual Property“(IP)-Bausteine zum Einsatz. Neben der getrennten Entwicklung verschiedener Komponenten und Einheiten ermöglicht der plattformbasierte Entwurf die Trennung der Funktionsblöcke von der eigentlichen Architektur des zusammengesetzten Systems, wie beispielsweise des SoCs.

Die Beschreibung der Hardwaremodule und Funktionsblöcke findet dabei in der Regel in einer Hardware-Beschreibungssprache wie z. B. VHDL oder Verilog statt. Hierbei werden sogenannte Virtuelle Komponenten erzeugt. Der Entwurf der Hardware in einer Hardware-Beschreibungssprache ermöglicht die Verifikation und das Testen unter realen Bedingungen, noch bevor der erste Wafer produziert wurde. Die Kosten für die Produktion eines solchen Wafers sind sehr hoch. Daher ist es entscheidend, dass bereits der erste Entwurf möglichst frei von Fehlern ist. Der plattformbasierte Entwurf bietet neben der Reduzierung des Aufwandes für Entwicklung und Verifikation die Wiederverwendbarkeit ganzer Subsysteme. Trotz dieser Vorteile werden besonders im Bereich der Automotive-SoC-Entwicklung weiterhin anwendungsspezifische und optimierte ICs zur Erreichung einer höheren Performance benötigt [14, p. 60].

2.6.7 Hardware/Software-Co-Entwurf

Da SoCs in der Regel heterogene Systeme darstellen, welche über einen oder mehrere Prozessoren und damit über Software verfügen, wird ein Hardware/Software-Co-Entwurf benötigt. Dabei werden Software und Hardware von weitestgehend getrennt voneinander arbeitenden Entwicklern/-innen entworfen. Zuvor muss eine Einteilung der funktionalen Module auf System-Ebene in die entsprechende Entwurfsdomäne erfolgen. Wie im vorangegangenen Abschnitt beschrieben, hängt die Softwareentwicklung dabei stark von der Hardware ab und kann meist erst zu einem späteren Zeitpunkt der Entwicklung starten. Für die parallele Entwicklung von Hardware und Software ist ein hohes Maß an Kommunikation zwischen Hardware- und Softwareentwicklern/-innen nötig. Eine gemeinsame Simulation ist in der Regel erst in einer späten Phase der Entwicklung möglich. Um die Software bereits vor der Produktion der Hardware gemeinsam mit dieser simulieren zu können, werden Prototypen noch vor der finalen Fertigstellung der Hardware genutzt. Dabei kann ein Prototyp der Hardware mittels FPGA oder die Hardware als Virtueller Prototyp implementiert

werden. Auf Virtuelle Prototypen wird im nachfolgenden Abschnitt näher eingegangen. Ein wichtiger Aspekt des Hardware/Software-Co-Entwurfes und der Simulation mittels Prototypen ist die damit verbundene frühzeitige Validierung der Hardware-Software-Schnittstelle.

Hardware und Software, welche fertig entwickelt wurden, werden zum Zeitpunkt der Systemintegration zusammengeführt und können anschließend als Gesamtsystem getestet und verifiziert werden [14, p. 37].

2.6.8 Verifikation und Validierung von Hardware-/Softwaresystemen

Die Verifikation ist ein unverzichtbarer Bestandteil des Entwurfs von Hardware/Software-Systemen, wie den Automotive SoCs. Daher werden sowohl Verifikation wie auch Validation in jedem Teil des Industriestandards ISO 26262 gefordert und sind essenziell, um für ein Produkt bzw. einen Prozess das höchste Maß an funktionaler Sicherheit und Qualität sicherzustellen [18]. Die Verifikation soll darüber hinaus, vereinfacht ausgedrückt, die Korrektheit eines Systems beweisen. Die Bedeutung der Korrektheit lässt sich dem IEEE-Standard 610.12 [30] nach jedoch unterschiedlich auslegen:

- Erfüllung der Kundenerwartungen unabhängig von der Spezifikation
- Fehlerfreiheit des Systems in Bezug auf Spezifikation und Entwurf
- Erfüllung der Systemanforderungen

Die Erfüllung der Kundenerwartungen, unabhängig von der Spezifikation, spielt bei der SoC-Entwicklung eher eine untergeordnete Rolle. SoCs werden oftmals kundenspezifisch entwickelt, und die Entwicklung des SoCs basiert dabei, wie in Abschnitt 3.1.1 beschrieben, auf einer vertraglichen Vereinbarung, welche dem Lasten- und Pflichtenheft zugrunde liegt. Somit wird ein Entwicklungsvorhaben vereinbart, in dem lediglich zugesichert wird, was in Lastenheft und Pflichtenheft spezifiziert wurde [2, p. 4].

Bei der Entwicklung eines SoCs gibt es eine Vielzahl unterschiedlicher Fehlerpotenziale und Fehlerarten über den gesamten Entwicklungs-Flow hinweg. Hierbei lassen sich im Kontext der Verifikation zwei Arten von Fehlern definieren:

- Ein **Spezifikationsfehler** ist die falsche, nicht eindeutig oder nicht vollständige Spezifizierung einer Systemanforderung bzw. Information in der Spezifikation.

- Bei **Entwurfsfehlern** handelt es sich um eine falsche Implementierung im Sinne einer Abweichung von der Spezifikation [2, p. 5].

Beispielsweise kann eine fehlende Eindeutigkeit bei den zu Beginn der Entwicklung spezifizierten Kundenanforderungen zu einer falschen Interpretation durch die Entwickler/-innen führen, welche darauf aufbauend Entwurfsentscheidungen treffen und Systemanforderungen in den Spezifikationsdokumenten schreiben. Somit wird das System im Sinne der Systemanforderungen falsch spezifiziert. Eine fehlende Eindeutigkeit der Systemanforderungen, welche sich z. B. aus der Beschreibung mittels natürlicher Sprache ergeben kann, führt dagegen dazu, dass die Entwickler/-innen gezwungen sind, zu interpretieren. Somit entsteht ein eigenes Bild des Systems in der Vorstellung der Entwickler/-innen, welches nicht mehr der Spezifikation entspricht und falsch interpretiert sein könnte. Implementieren die Entwickler/-innen basierend auf diesem eigenen Bild des Systems, entstehen Entwurfsfehler, da der Entwurf nicht der Spezifikation entspricht. Um das Risiko einer fehlerhaften Implementierung aufgrund von Interpretationen zu minimieren, ist es wichtig, dass ein nicht am Entwurf beteiligter Entwickler die Verifikation des Systems durchführt.

Eine nicht vollständige Spezifikation birgt besonders schwerwiegende Gefahren, da so Szenarien, welche in der Nutzung des Systems auftreten können, nicht spezifiziert und damit auch nicht verifiziert werden. Es besteht somit die Gefahr eines unvorhersehbaren Zustandes bzw. Verhaltens und damit im schlimmsten Fall einer Gefährdung von Menschenleben, wie am Beispiel der Fahrassistenzsysteme deutlich wird. Spezifikationsfehler müssen daher mittels umfangreicher Validierung der Spezifikation minimiert werden.

Die Verifikation weist nach, dass ein System bezogen auf seine Spezifikation korrekt entworfen wurde. Es kann somit der Nachweis erbracht werden, dass das System sowohl die funktionalen Teile als auch Teile der nichtfunktionalen Anforderungen der Spezifikation erfüllt. Befinden sich jedoch Fehler in der Spezifikation oder ist die Spezifikation nicht vollständig, gilt der Entwurf dennoch als korrekt im Sinne der Verifikation.

Neben der bereits erwähnten Problematik von Interpretationen beim Entwurf eines Systems aufgrund einer nicht eindeutigen Spezifizierung gilt dieselbe Problematik für die Verifikation. Ist die Spezifikation nicht eindeutig, sind die Verifikationsingenieure/-innen gezwungen, die Verifikation aufgrund von Interpretationen zu planen und zu entwickeln. So entsteht auch in diesem Fall ein eigenes Bild des Systems in der Vorstellung der Verifikationsingenieure/-innen [2, pp. 11–12]. Darüber hinaus besteht das Risiko, dass die Verifikationsingenieure/-innen sich die fehlenden Informationen der Spezifikation durch einen Blick in

den Systementwurf einholen und so die Verifikation auf Basis des implementierten Entwurfs entsteht. Ein solches Vorgehen würde jedoch die Verifikation im Regelfall entkräften, da so der Entwurf nicht mehr gegen die Spezifikation, sondern gegen sich selbst geprüft wird und damit das Aufdecken von Fehlern im Entwurf mittels Verifikation unwahrscheinlich wird.

2.7 Modellgetriebener Systementwurf

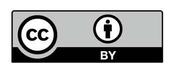
Der in Abschnitt 2.5 vorgestellte Ansatz der modellbasierten Systementwicklung dient in erster Linie zur modellbasierten Konzeptionierung und Spezifikation des Systems. Jedoch müssen auch bei der modellbasierten Systementwicklung die Informationen aus der modellierten Spezifikation – zum Beispiel für den Softwareentwurf – in eine höhere Programmiersprache transformiert werden. Gleiches gilt für die Hardware; auch hier müssen die Informationen in eine Hardware-Beschreibungssprache übersetzt werden. Dies führt zu einem erheblichen Aufwand für die Übertragung bereits beschriebener Informationen und ist zu dem sehr fehleranfällig.

Der modellgetriebene Systementwurf kann als Erweiterung der modellbasierten Systementwicklung gesehen werden. Hierbei wird auf Grundlage der modellierten Spezifikation die Automatisierung der Entwurfsimplementierung ermöglicht und somit der Aufwand für den Systementwurf reduziert. Das bedeutet, es wird der Code von Teilen oder eines ganzen Systems auf Grundlage von Modellen generiert.

Hierzu bedarf es einer maschinenlesbaren Sprache zur Beschreibung des Systems im Modell. Die Verwendung einer universellen und meist formalisierten Modellierungssprache wie der UML bietet die Möglichkeit, auch über Entwicklungsteams hinweg kommunizieren zu können [22]. Reicht diese universelle Sprache nicht aus, kommen DSML zum Einsatz. Um diese maschinenlesbaren Modelle in Code zu transformieren, werden Codegeneratoren benötigt, welche ein Mapping zwischen Modellelementen und Code-Frame beinhalten.

Der Ansatz des modellgetriebenen Systementwurfs führt neben der Reduzierung des Aufwandes zu einer Steigerung der Konsistenz zwischen Implementierung und Spezifikation. In aktuellen Entwicklungsprojekten werden Spezifikation und Implementierung getrennt voneinander gespeichert und gepflegt. Das führt dazu, dass Entwurfsentscheidungen, die während der Implementierung getroffen werden, sowie Anpassungen, die vorgenommen werden, nicht auch in der Entwurfsspezifikation dokumentiert werden. Somit kann es zu Inkonsistenzen und damit beispielsweise zu Problemen bei der Verifikation kommen. Erfolgt

der Entwurf jedoch mittels modellgetriebener Synthese und ist das Systemmodell Teil der Spezifikation, sind Implementierung und Spezifikation zu jeder Zeit konsistent [31].





3 Stand der Technik

Im vorangegangenen Kapitel wurden die Grundlagen der SoC-Entwicklung sowie Themen aus der modellbasierten Systementwicklung behandelt. Als Teil dessen, wurde der SoC-Entwicklungs-Flow mit seinen unterschiedlichen Phasen der Entwicklung vorgestellt. Die in dieser Arbeit entwickelte modellbasierte Entwicklungsmethode soll, wie in Kapitel 1 beschrieben, über alle Phasen der Entwicklung hinweg durch einen modellbasierten Ansatz Defizite im aktuellen Flow beseitigen und die Effizienz der SoC-Entwicklung in Gesamtheit steigern. Eine detaillierte Behandlung der Methode folgt in Kapitel 4. Nach aktuellem Stand gibt es nach Wissen des Autors keine existierende Lösung für die modellbasierte Systementwicklung von Automotive SoCs, welche dabei den gesamten Entwicklungs-Flow von der Analyse der Kundenanforderungen bis zum Systementwurf berücksichtigt und dabei eine Automatisierung des Entwurfs sowie der Verifikationserstellung ermöglicht. Daher erfolgt in diesem Kapitel eine Betrachtung von Lösungsansätzen aus der Literatur, welche lediglich Teilbereiche des SoC-Entwicklungs-Flows abdecken bzw. einen Teil der gestellten Anforderungen erfüllen. In dem folgenden Kapitel werden daher die existierenden Forschungs- und Lösungsansätze auf die Erfüllung der in Abschnitt 1.1 definierten Anforderungen **A1** (Reduzierung von nachträglichen Änderungen), **A2** (Reduzierung von Implementierungsfehlern), **A3** (Reduzieren von Inkonsistenzen) sowie auf die Erfüllung der Anforderungen **A7** (Reduzierung Aufwand Entwurf) und **A8** (Reduzierung Aufwand Verifikation) untersucht. Die Anforderungen **A4** (Integrierbarkeit der Methode in bestehenden Flow), **A5** (Einfluss der Methode auf Spezifikationsaufwand) und **A6** (Anwenderfreundlichkeit) beziehen sich auf die Gesamtlösung dieser Arbeit. Deren Erfüllung wurde in den hier diskutierten Teillösungen der Literatur nicht ausreichend betrachtet. Aus diesem Grund lassen sich die Arbeiten nur bedingt auf die Erfüllung dieser Anforderungen überprüfen.

A. Kirchner, *Entwicklung von Methoden zur abstrakten Modellierung von Automotive Systems-on-Chips*, https://doi.org/10.1007/978-3-658-38437-1_3

Die Erfüllung der Anforderungen **A1** (Reduzierung von nachträglichen Änderungen), **A2** (Reduzierung von Implementierungsfehlern) und **A3** (Reduzieren von Inkonsistenzen) wird, wie nachfolgend in Kapitel 4 beschrieben, in der hier vorliegenden Arbeit durch die Definition einer Modellierungsmethode gepaart mit einer Arbeitsweise nach dem Top-down-Ansatz und der Formalisierung der modellierten Spezifikation für den SoC-Bereich erreicht. Daher werden in Abschnitt 3.1 Lösungen aus der Literatur für die folgenden Themengebiete diskutiert:

- Die **modellbasierte Systementwicklung**, welche allgemeine Modellierungsmethoden und modellierungssprachenbasierte Lösungen für die modellbasierte Entwicklung bietet
- Die **Formalisierung und Modellierung der Spezifikation**, welche sich meist auf einen bestimmten Anwendungsbereich fokussieren

Die Anforderungen **A7** (Reduzierung Aufwand Entwurf) und **A8** (Reduzierung Aufwand Verifikation) stehen für die Effizienzsteigerung in der Entwurfsphase und der daran anschließenden Verifikation. In Abschnitt 3.2 folgen daher:

- Lösungen aus der Literatur für den **modellgetriebenen Systementwurf sowie die Entwicklung von Virtuellen Prototypen** aus dem Bereich der Eingebetteten Systeme.
- Bestehende Lösungen für **die Reduzierung des Aufwandes bei der Verifikationserstellung**.

Eine Abgrenzung der Arbeit gegen die hier untersuchten bestehenden Lösungen aus der Literatur folgt in Abschnitt 3.3.

3.1 Modellbasierte Systementwicklung

Die Behebung der Defizite in der aktuellen Spezifikationsmethodik ist in Abschnitt 1.1 durch die Definition der Anforderungen **A1**, **A2** und **A3** abgebildet. Mittels einer modellbasierten Entwicklungsmethode, welche zugleich eine Arbeitsweise für die Systementwicklung beinhaltet, kann die Analyse der Anforderungen und die Strukturierung des Modells unterstützt und so nachträgliche Änderungen (**A1**) reduziert werden. Aus diesem Grund betrachtet der nachfolgende Abschnitt Modellierungsmethoden aus der Literatur für die Systementwicklung.

Für die Reduktion von Inkonsistenzen und Implementierungsfehlern, welche aus der Spezifikation resultieren, ist die Formalisierung der Spezifikationssprache jedoch zusätzlich nötig, da diese die Eindeutigkeit der Spezifikation maßgeblich erhöht. Die modellbasierte Entwicklungsmethode dieser Arbeit soll über alle in Abschnitt 2.6 dargestellten Phasen hinweg eine modellbasierte Entwicklung und Spezifikation in der SoC-Entwicklung ermöglichen.

3.1.1 Modellierungsmethoden

Die Literatur bietet verschiedene Modellierungsmethoden für die Systementwicklung an, welche sich auf die Entwicklung von Eingebetteten Systeme anwenden lassen. Die in [32] beschriebene Modellierungsmethode der OMG (Object Management Group) trägt die Bezeichnung Model Driven Architecture (MDA). Hierbei handelt es sich um einen generellen Modellierungsansatz, welcher sich domänen- und modellierungssprachen-übergreifend anwenden lässt. Als Teil des Ansatzes werden unter Verwendung von Industriestandards die generelle Struktur, Notation und Semantik für die Modellierung definiert. Einen vergleichbaren Ansatz bietet die Modellierungsmethode in [33], genannt Electronic System Level (ESL)-Design. Diese Methode wurde darüber hinaus jedoch speziell für die Modellierung und Entwicklung von SoCs entwickelt und bietet hierfür eine Top-down-basierte Arbeitsweise. Ein wichtiger Aspekt der Methode ist die modellbasierte Analyse und Simulation des abstrahierten Entwurfs in einer frühen Phase der Entwicklung.

Die in [22] vorgestellte SYSMOD-Methode sowie die in [34] vorgestellte SPES-Modellierungsmethode bietet ebenfalls einen Ansatz für die Systemmodellierung. Jedoch orientieren sich beide Methoden in der Umsetzung der Modellierungsschritte an den in der UML bzw. SysML enthaltenen Diagrammen. Zusätzlich zur reinen Modellierungsmethode definiert die SPES2020-Methode aus [34] eine spezielle Modellstruktur. So bestehen heutige Systeme in der Regel aus einzelnen Subsystemen, welche wiederum aus Subsystemen oder einzelnen Modulen und Submodulen bestehen. Diese Subsysteme werden oftmals mehr oder weniger getrennt entwickelt, wodurch ebenso mehrere unabhängige Modelle entstehen. Um die so voneinander unabhängig entstehenden Modelle unterschiedlicher Subsysteme zusammensetzen zu können, werden als Teil der SPES2020-Methode eine Arbeitsweise für die Modellierung sowie Teile der Struktur formalisiert. Dabei dient die SPES2020-Methode in erster Linie zur Modellierung von großen Gesamtsystemen auf abstrakter Ebene, ohne dabei die spezifischen Anforderungen einer Domäne oder der SoC-Entwicklung zu erfüllen.

Die in diesem Abschnitt vorgestellten Modellierungsmethoden bieten Lösungen für die Modellierung und modellbasierte Entwicklung von Systemen. Dabei dient die Modellierung des Systems jedoch in der Regel als Ergänzung für die aktuell genutzten Beschreibungsformen und ist nicht als Teil der Spezifikation vorgesehen. Darüber hinaus bleibt bei der Anwendung einer Modellierungsmethode ohne die Verwendung einer eindeutigen Beschreibung die Interpretierbarkeit und damit die fehlende Eindeutigkeit der Beschreibung weitestgehend bestehen. Dadurch bleibt auch das Risiko für Implementierungsfehler aufgrund einer unzureichenden Spezifikation bestehen und ebenso das Risiko für Inkonsistenzen zwischen implementiertem Entwurf und Spezifikation. Aus diesem Grund werden im nachfolgenden Abschnitt Lösungen aus der Literatur für die modellbasierte Spezifikation und Formalisierung der Spezifikation betrachtet.

3.1.2 Methoden zur modellbasierten Spezifikation und Formalisierung

Die Grundlagen der modellbasierten Systementwicklung sowie die Formalisierung der Spezifikation wurden in Abschnitt 2.5 behandelt. Mögliche Lösungen für die Formalisierung und Modellierung der Spezifikation aus der Literatur werden im Nachfolgenden diskutiert. Dabei werden ebenfalls Ansätze diskutiert, welche lediglich auf eine einzelne Domäne der SoC-Entwicklung anwendbar sind. Der Abschnitt ist in drei Unterabschnitte unterteilt: Spezifikationslogiken, objektorientierte Modellierung und aufgrund der besonderen Nähe zu der in dieser Arbeit vorgestellten Methode, davon losgelöst, SysML-basierte Modellierungsansätze.

Spezifikationslogiken

Die Literatur bietet neben den gängigen grafischen Modellierungssprachen auch Term-basierte Modellierungssprachen für die Analyse und Beschreibung von Systemen. So bieten die Arbeiten [35], [36] und [37] Modellierungssprachen zur Spezifikation von Eingebetteten Systemen, welche auf Termen und Algorithmen basieren. In [35] findet dabei zu Beginn des Entwicklungs-Flows eine Dekomposition der Systemfunktionalität mittels Termen statt, um diese anschließend in die Domänen Analog- und Digitaltechnik aufzuteilen. Auf Grundlage der so erhaltenen Gruppierungen der Funktionalitäten werden diese den jeweiligen Modulen des Systems zugeordnet. Die Steigerung der Effizienz in der Verifikation sowie die Formalisierung der Spezifikation sind Ziele der in [37] beschriebenen Methode. Dabei wird zur Beschreibung des Systems die Projection Temporal Logic (PTL) verwendet, um das Verhalten des Systems als diskrete Zustandsfolgen zu spezifizieren.

Darauf aufbauend nutzt der Autor die Programmiersprache Tempura, um eine Ausführbarkeit der Spezifikation zu erreichen. Die Autoren/-innen in [36] definieren hingegen eine Sprache basierend auf einer typisierten Logik erster Ordnung. Dabei wird das System als eine Sammlung aus verschiedenen Konzepten beschrieben und somit der Zustandsraum des Systems modelliert. Ein Konzept beschreibt dabei einen Teilbereich des Modells. Term-basierte Sprachen lassen sich aufgrund ihrer Nähe zur Mathematik leichter formalisieren, bergen jedoch viele Nachteile bei der Lesbarkeit und Fehlerwahrscheinlichkeit, wodurch die Anwenderfreundlichkeit – verglichen mit grafischen Modellierungssprachen – sinkt

Objektorientierte Modellierungssprachen

Weitaus verbreiteter als die Spezifikationslogiken ist in der Entwicklung von Eingebetteten Systemen die Verwendung von objektorientierten Modellierungssprachen. Ein Auszug der verfügbaren grafischen Modellierungssprachen bietet der Abschnitt 2.5. Die in Arbeit [38] entwickelte Methode beschäftigt sich nicht in erster Linie mit der Modellierung der Spezifikation, sondern bietet einen Ansatz zur Übersetzung von in natürlicher Sprache geschriebenen Softwareanforderungen in ein UML-Modell. Dazu werden die Anforderungen, welche im Concern-Aware Requirements Engineering (CARE)-Format spezifiziert wurden, in UML-Klassendiagramme übersetzt, um diese für die Generierung weiterverwenden zu können. Die Literaturquellen [39], [40] und [41] bieten dementgegen Ansätze zur rein modellbasierten Entwicklung. So wird eine Teilmenge der UML, bestehend aus Klassen-, State- und Sequenzdiagrammen, zur Spezifikation des Systems genutzt. Zusätzlich zur Verwendung einer standardisierten und maschinenlesbaren Modellierungssprache definiert der Autor eine UML Virtual Machine (UVM). Mithilfe dieser UVM lassen sich die State-Diagramme in Binärcode und die Sequenzdiagramme in Bytecode übersetzen und somit eine Ausführbarkeit der Spezifikation erreichen.

Einen semi-formalen Ansatz für die Modellierung der Anforderungen bietet die Arbeit in [42]. Dabei wird das in der SysML enthaltene Modellelement, genannt Requirements-Block, zur Definition von Anforderungs-Pattern genutzt. Dieser Ansatz ermöglicht die Vereinheitlichung und damit die Wiederverwendbarkeit von modellierten Anforderungen in der SoC-Entwicklung. Zusätzlich zur Aufwandsreduzierung werden die Anwender/-innen durch die Definition von Pattern bei der korrekten Beschreibung der Anforderungen unterstützt.

Der in [43] behandelte Ansatz kombiniert UML und „Actor-oriented Modelling“ zur modellbasierten Spezifikation von SoCs. „Actor-oriented Modelling“ bezeichnet eine Formalisierung von Techniken, welche über viele Jahre hinweg im Entwurf Eingebetteter Systeme eingesetzt wurden. Dabei liegt das Hauptmerkmal

des „Actor-oriented Modelling“-Ansatzes, anders als bei der UML, auf der Modellierung von Verhalten zusammengesetzter Systeme. Die UML bietet hingegen die Darstellung verschiedener Aspekte in einzelnen Diagrammen.

Die Arbeiten [44] und [45] ermöglichen die Entwicklung von „Realtime and Embedded Systems“ (RTES) durch die Verwendung von MARTE, einem UML-Profil für die Modellierung und Analyse von RTES. Hierbei verwenden die Autoren/-innen in [44] zusätzlich eine Kombination aus MARTE und SPES, um die Architektur von RTES mittels verschiedener „Viewpoints“ (Sichtweisen) abzubilden. Dabei werden die einzelnen Sichtweisen der SPES-Methode auf die MARTE-Semantik zugeordnet.

Einen vergleichbaren Ansatz bietet die Arbeit [46]. Hierbei wird auf Basis der UML und des MARTE-Profils eine „Single-Source“-Modellierungsmethode (aus einer Hand) für die Modellierung von Eingebetteten Hardware-/Softwaresystemen vorgestellt. Dabei ermöglicht die Methode die Trennung verschiedener Aspekte der Systementwicklung, die inkrementelle Modellierung und die Modellierung funktionaler Komponenten. Sie unterstützt darüber hinaus wichtige Entwurfsaufgaben wie Design Space Exploration und die Software-Synthese.

Die Publikation [47] schlägt ein Hardware/Software-Co-Entwurf-Framework vor, welches die Inkompatibilität über Hardware-Software-Grenzen hinweg beheben und somit Schwierigkeiten bei der Verifizierung reduzieren soll. Dabei wird eine SystemC-basierte Modellierung in Form eines Model-of-Computation (MoC, Berechnungsmodell) sowie eines Model-of-Communication (MoComm, Kommunikationsmodell) zur Beschreibung des Systems verwendet. Ein MoC wird dabei zur Modellierung von Operationen und Berechnungen verwendet, sowohl für Software als auch für Hardware. Mithilfe des MoComm wird die Datenkommunikation zwischen einzelnen Hardware MoCs modelliert.

[48] nutzt die UML als Frontend zur grafischen Modellierung einer IP-XACT-basierten IP-Beschreibung. Hierzu wird in der Arbeit ein Profil der UML vorgestellt sowie eine Bibliothek definiert, welche eine Zuordnung zwischen IP-XACT und UML-Elementen ermöglicht. Durch diesen Ansatz wird die Anwenderfreundlichkeit der grafischen Beschreibung in der UML mit den Stärken der IP-XACT im Bereich Austauschbarkeit und Weiterverwendbarkeit kombiniert.

Neben den methodischen Lösungen aus dem akademischen Bereich bietet beispielsweise das grafische Modellierungswerkzeug **STATEMATE** der Firma *I-Logix* die entsprechende Tool-Umgebung für die Entwicklung Eingebetteter Systeme. Dabei fokussiert sich das Werkzeug auf die Beschreibung von Systemen mittels Zustandsautomaten. Es wurde bereits vor der UML entwickelt und lässt sich als Vorgänger des Modellierungswerkzeugs **IBM® Rational® Rhapsody®** verstehen [22].

SysML-basierte Modellierung

Da die in dieser Arbeit entwickelte Methode auf der SysML als Modellierungssprache aufbaut, sollen im folgenden Abschnitt entsprechende SysML-basierte Lösungen aus der Literatur gesondert betrachtet werden. So behandelt die Veröffentlichung [49] die Defizite bei der Dokumentation von Anforderungen in natürlicher Sprache und nutzt zusätzlich zum Use Case Diagram das in SysML enthaltene Requirement Diagram, um die Beziehungen der einzelnen Anforderungen zueinander im Modell abbilden zu können. Des Weiteren wird als zusätzliche Darstellungsform der SysML die tabellarische Darstellung der Anforderungen zur Steigerung der Übersichtlichkeit vorgeschlagen. Die Modellierung der Anforderungen soll hierbei noch vor der Anwendungsfallanalyse im Use Case Diagram stattfinden. Einen vergleichbaren Ansatz verfolgen die Autoren/-innen in [50]. Hier wird eine Kombination aus UML-, SysML- und MARTE-Stereotypen verwendet, um das Anforderungsmanagement in der Domäne Eingebetteter Systeme zu verbessern. Die UML dient dabei zur Modellierung der Funktionalen-, der Strukturellen- sowie der Verhaltenssicht des Systems. Die SysML wird als Ergänzung verwendet, um die Anforderungen im Requirement Diagram und die Beziehungen untereinander abzubilden. Zusätzlich wird eine „Traceability"-Matrix vorgestellt, mit der sich das Anforderungsmanagement optimieren lässt. Die Stereotypen des MARTE-Profils werden dabei verwendet, um die nichtfunktionalen Anforderungen zu spezifizieren.

Die Publikation in [51] kommt aus dem Umfeld der Chemie, beschäftigt sich jedoch ebenso mit der Beschreibung eines Systems mittels SysML. Dabei wendet der Autor zudem die SYSMOD-Methode für die Modellierung an. Ziel der Arbeit ist es, den Nutzen der SysML-basierten Systembeschreibung für die Spezifikation von Systemen aus dem Chemielabor zu testen, um daraus Erkenntnisse über die Funktionsweise des Systems gewinnen zu können.

Eine Fallstudie zum Nutzen von UML 2.0 und SysML für die SoC-Entwicklung wurde in [52] anhand eines Wireless LAN SoCs durchgeführt. Dabei kommen die Autoren/-innen zu dem Schluss, dass durch die Entwicklung der UML 2.0 und der SysML ein großer Meilenstein für die SoC-Entwicklung erreicht wurde, darüber hinaus jedoch die Standardisierung und Formalisierung der Sprachen erforderlich ist. Zudem wird die Notwendigkeit von leistungsbezogenen Aspekten für die Modellierung von SoCs genannt. Durch die Erweiterung der SysML und die Kombination von SysML mit MATLAB-Simulink-Modellen ermöglicht [53] die Modellierung von kontinuierlichem Zeitverhalten. Zusätzlich wird durch die Entwicklung eines Eclipse-basierten Ausführungstools die Co-Simulation der beiden Modelle ermöglicht [54] [55].

Für die Modellierung und modellbasierte Entwicklung von Systemen mittels SysML und UML steht eine Vielzahl unterschiedlicher Modellierungswerkzeuge zur Verfügung. So bietet die Firma *Sparx Systems* das Werkzeug **Enterprise Architect,** welches unter anderem die Anwender/-innen bei der Modellierung von UML und SysML unterstützt. Neben der reinen Modellierung bietet Enterprise Architect die Möglichkeit der Simulation sowie verschiedene Generatoren für die modellgetriebene Entwicklung [56].

VisualSim Architect der Firma *Mirabilis Design* ist ein weiteres Modellierungswerkzeug, welches für die Modellierung mit SysML entwickelt wurde. Das Modellierungswerkzeug ist dabei Teil einer Tool-Umgebung namens VisualSim. Mithilfe der darin enthaltenen Lösungen soll die Brücke zwischen SysML-basierter Spezifikation und MATLAB-Simulink-programmierter Implementierung geschlossen werden [57].

Neben den konventionellen Werkzeugen bietet die Open-Source-Lösung **Eclipse Papyrus** die modellbasierte Systementwicklung und Modellierung. Es bietet neben anderen Standard-Modellierungssprachen die Möglichkeit der Modellierung von UML und SysML und kann durch seinen Open-Source-Ansatz leicht an die jeweilige Domäne angepasst werden. Darüber hinaus wurde das „Papyrus Industry Consortium“ gegründet, um die Bedürfnisse der Industrie bei der Entwicklung des Werkzeugs zu berücksichtigen [58].

Eine weitere Entwicklungs- und Modellierungsumgebung bietet das in dieser Arbeit verwendete Werkzeug **IBM® Rational® Rhapsody®** der Firma *IBM*. Es bildet ein zentrales Werkzeug der Rational-Tool-Umgebung und bietet neben anderen Modellierungssprachen eine Oberfläche für die Modellierung von UML und SysML [59]. Eine nähere Beschreibung des Werkzeugs erfolgt in Abschnitt 5.1.

Abgrenzung zur Arbeit

Die in Abschnitt 3.1.1 vorgestellten Modellierungsmethoden bieten Lösungen für die modellbasierte Systementwicklung. Dabei wird als Teil der Modellierungsmethoden eine Arbeitsweise implementiert, welche die modellbasierte Analyse der Kundenanforderungen und der Anwendungsfälle beinhaltet. Zusätzlich ermöglicht die Modellierung des Systems ein besseres Systemverständnis sowie eine bessere Kommunikation zwischen Kunde und Entwickler und ermöglicht damit die Reduzierung von nachträglichen Änderungen an den vertraglich vereinbarten Anforderungen (**A1**).

Abschnitt 3.1.2 diskutiert Lösungen für die Modellierung von Systemen und die Formalisierung der Spezifikation. Dabei kann durch die Modellierung und im besonderen Maße durch die Formalisierung der Spezifikation die Eindeutigkeit dieser erheblich gesteigert und somit das Risiko für Interpretationen minimiert werden.

Somit kann die Reduzierung von Implementierungsfehlern aufgrund einer unzureichenden Spezifikation (**A2**) und auch das Risiko von Inkonsistenzen (**A3**) durch die hier gezeigten Lösungen aus der Literatur erreicht werden. Dadurch ist zudem eine indirekte Aufwandsreduzierung, bezogen auf das Beheben von Fehlern und das Durchführen von Nachbearbeitungen, möglich. Der Aufwand bei der Implementierung des Entwurfs selbst (**A7**) kann durch diese Lösungen nicht reduziert werden. Ebenso kann für die Verifikation der Aufwand nur indirekt durch die höhere Eindeutigkeit der Spezifikation erreicht werden. Der Aufwand für die Erstellung der Verifikation bleibt unverändert hoch (**A8**). Einige der hier vorgestellten Lösungen kombinieren die Modellierung der Spezifikation mit einer der zuvor behandelten Modellierungsmethoden und ermöglichen somit sowohl die teilweise Erfüllung der Anforderungen **A1** und **A2** wie auch die Anforderung **A3,** wie in Tabelle 3.1 abgebildet.

3.2 Modellgetriebener Systementwurf und Verifikation

Neben der modellbasierten Spezifikation und Entwicklung von Systemen bietet die Literatur Lösungen der darauf aufbauenden modellgetriebenen Automatisierung des Systementwurfs und damit verbunden der Entwicklung von Virtuellen Prototypen. Der modellgetriebene Systementwurf und die damit verbundene Generierung finden dabei in verschiedenen Domänen Anwendung. Im nachfolgenden Abschnitt werden aufgrund der besonderen Anforderungen der Entwicklung von Eingebetteten Systemen in erster Linie Ansätze aus diesem Bereich diskutiert. Dabei wird der Einfluss der Lösungen auf den Implementierungsaufwand und damit die Erfüllung der Anforderungen **A7** diskutiert.

Automatisierung Systementwurf

Die Publikationen [60] und [61] beschreiben einen Ansatz zur SysML-basierten Modellierung und anschließender Codegenerierung. Dabei findet mittels SysML-Profilen ein Mapping zwischen SysML-Stereotypen und SystemC-Code-Elementen statt. Für die automatisierte Codegenerierung verwendet der Autor die in Atego Artisan Studio enthaltenen Frameworks für die Model-zu-Code-Generierung TDK (Template Development Kit) und Shadow ACS. Ziel der Arbeiten ist es, die Lücke zwischen UML-/SysML-basierter Beschreibung und SystemC-basierter Verifikation und Synthese zu schließen. **PTC Integrity Modeler**, ehemals **Atego Artisan Studio** der Firma *PTC,* ist ein Werkzeug für die Modellierung von SysML, UML sowie für die Variabilitätsmodellierung [62] [63].

Einen vergleichbaren Ansatz bietet die Arbeit [64]; hierbei findet die Generierung des SystemC-Codes in zwei Schritten statt. Im ersten Schritt werden die UML-Modelle in XMI-Dateien umgewandelt. Anschließend wird auf Basis von vordefinierten Templates SystemC-Code generiert. Die Besonderheit der Arbeit ist die Modellierung und Generierung eines Bussystems mittels UML State Chart Diagram [65].

Wie bereits erwähnt ist die Anwendung von Virtuellen Prototypen weitestgehend etabliert. Darüber hinaus wurden bereits erste Möglichkeiten der Automatisierung entwickelt. So verwendet der Autor in [66] den generierten SystemC-Code zur Realisierung eines Virtuellen Prototyps. Dabei dient das SysML-Modell als Startpunkt der Entwicklung. Auf Basis des SysML-Modells wird eine grobe IP-XACT-Architekturbeschreibung erzeugt, welche im Nachfolgenden weiter verfeinert werden muss. Die Generierung des SystemC-Codes erfolgt somit auf Grundlage der verfeinerten IP-XACT-Registerbeschreibung.

Die Arbeit in [67] nutzt den modellbasierten Ansatz ebenfalls, um die Effizienz der Entwicklung von Prototypen zu steigern. Dabei wird die Architektur von SIMD (Single Instruction Multiple Data) SOCs mittels Kombination aus UML2 und MARTE-Profil beschrieben. Die so erhaltene Architekturbeschreibung wird um Bestandteile bestehender IP-Bibliotheken und Instruktionen aus einem parallel spezifizierten Befehlssatz erweitert. Dies ermöglicht die Generierung von VHDL-Code mittels MARTE2VHDL-Transformation und bietet somit die Möglichkeit, verschiedene Konfigurationen eines Systems je nach Anwendungsfall zu erzeugen.

Entwicklung Virtueller Prototypen

Die vorliegende Arbeit baut hinsichtlich der Automatisierung bei der Entwicklung von Virtuellen Prototypen auf der Arbeit von [15] auf. Daher soll diese nachfolgend ausführlicher als eine Lösung aus der Literatur für die modellgetriebene Entwicklung von Virtuellen Prototypen behandelt werden. Die Arbeit konzentriert sich dabei auf die Generierung von SystemC-Code für die signalverarbeitenden Komponenten des SoCs. Die nachfolgende Abbildung 3.1 zeigt somit den aktuellen Stand der VP-Entwicklung auf den die in Abschnitt 4.3 beschriebene Generierung aufbaut.

Ausgangspunkt in der vorangegangenen Arbeit von [15] ist die Konzeptspezifikation, welche in der Regel in natürlicher Sprache beschrieben ist. Basierend auf der Konzeptspezifikation findet eine Verfeinerung der Informationen statt, wodurch das Modell der Signalverarbeitung, eine Registerbeschreibung in IP-XACT sowie die Entwurfsspezifikation entsteht. Aufbauend auf diesen drei Beschreibungsformen entstehen drei Pfade.

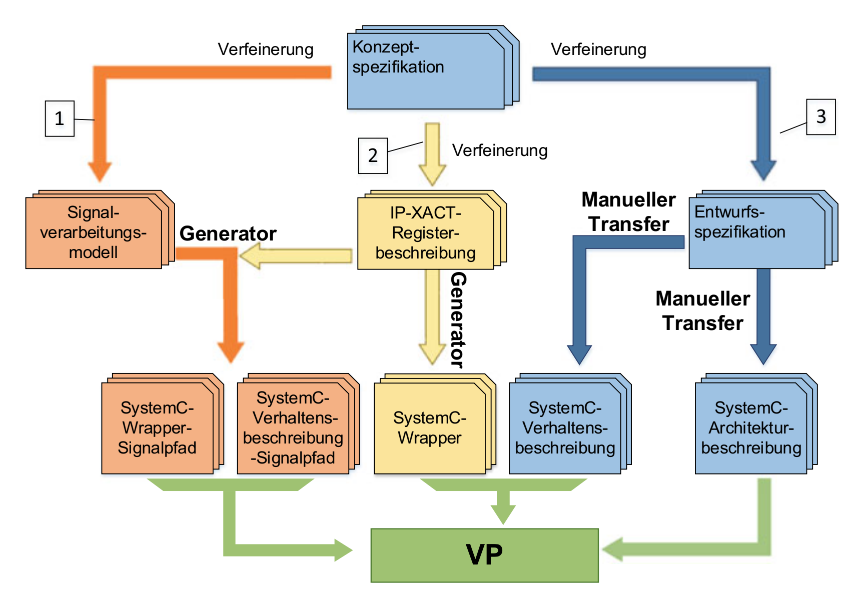


Abbildung 3.1 Aktueller Flow für Virtuelle Prototypen

Die Arbeit von [15] ermöglicht eine Generierung der signalverarbeitenden Komponenten, welche sich für den Virtuellen Prototyp aus einem sogenannten *SystemC-Wrapper-Signalpfad* und einer *SystemC-Verhaltensbeschreibung-Signalpfad* zusammengesetzt werden. Dabei wird zuerst ein Modell der Signalverarbeitung erstellt und mit den Informationen der Registerbeschreibung in IP-XACT kombiniert. SystemC-Wrapper lassen sich vereinfacht als leere Modulhüllen mit definierten Schnittstellen verstehen.

Für die übrigen kontrollflussorientierten Module ermöglicht die Arbeit zudem, wie im mittleren *Pfad 2* dargestellt, eine Generierung der SystemC-Wrapper auf Basis der IP-XACT-Registerbeschreibung. Um den generierten Wrapper jedoch mit der zugehörigen Verhaltensbeschreibung zu kombinieren, muss hier, wie im *Pfad 2* dargestellt, die *SystemC-Verhaltensbeschreibung* der kontrollflussorientierten Module manuell implementiert werden. Gleiches gilt für die Informationen der Architektur, welche ebenfalls manuell als Teil von *Pfad 3* aus den Anforderungen übertragen werden müssen. Es handelt sich somit in dieser Arbeit um einen teilautomatisierten Flow, der jedoch besonders in *Pfad 2* der kontrollflussorientierten Module Lücken aufweist.

Eine weitverbreitete Entwurfsumgebung mit Fokus auf die Signalverarbeitung sind **MATLAB** und **Simulink** der Firma *The MathWorks*. **MATLAB** bildet dabei die proprietäre Entwicklungsumgebung und Programmiersprache für die Visualisierung, während **Simulink** eine blockbasierte Modellierungs- und Simulationsoberfläche bietet. Vergleichbares bietet die objektorientierte Sprache **Modelica** der *Modelica Association,* für deren Nutzung verschiedene kommerzielle sowie Open-Source-Modellierungs- und Simulationswerkzeuge wie zum Beispiel **OpenModelica** zur Verfügung stehen [68] [69].

Die große Stärke von **MATLAB**, **Simulink** und **Modelica** liegt in der Simulation; daher bieten diese sich als Ergänzung zu SysML-Modellierungswerkzeugen an, welche wiederum in der Regel Vorteile bei der Anforderungsmodellierung und dem Gesamtsystemdesign bieten [54] [55] [22].

Für die Entwicklung, Verwaltung und Simulation von Virtuellen Prototypen bietet das Unternehmen *Synopsys* neben anderen Werkzeugen die Eclipse-basierte Entwicklungsumgebung **Virtualizer Studio,** welche eine Kombination verschiedener Modellierungs- und Debuggingwerkzeuge darstellt. Die Entwicklungsumgebung wurde dabei speziell für die Halbleitertechnik entwickelt [70].

Ergänzend bietet *Synopsys* für die Entwicklung von Virtuellen Prototypen das Werkzeug **Platform Architect Ultra** an, welches der frühzeitigen Analyse und Optimierung von Multicore-SoC-Architekturen hinsichtlich Performance und Power dient [71]. Weitere Werkzeuge für die Entwicklung und Simulation von Virtuellen Prototypen bieten die Firma *Mentor Graphics*, genannt **Vista Architect** [72], und die Firma *Cadence*, genannt **Virtual System Platform,** an [73].

Automatisierung Verifikationserstellung

Neben der Reduzierung des Aufwandes bei der Implementierung wurde als Ziel die Aufwandsreduzierung bei der Verifikation (**A8**) angestrebt. In der Literatur bestehen dazu verschiedene Lösungsansätze:

Die Arbeit [74] ermöglicht eine semi-formale Beschreibung der Spezifikation in tabellarischer Form, wodurch zum einen die Eindeutigkeit der Spezifikation erhöht werden kann und zudem die Validierung der Spezifikation erleichtert wird. Darauf aufbauend wird die Generierung von SystemVerilog-Properties für die Verifikation auf Basis der tabellarischen Spezifikation ermöglicht.

Einen vergleichbaren Lösungsansatz bietet die Arbeit [75]. Hierbei wird die Generierung von SystemVerilog-Assertions auf Basis von Prozessor-Architektur-Beschreibungen sowie von Modellen ermöglicht. Darüber hinaus streben beide Arbeiten die Formalisierung der Spezifikation an und ermöglichen somit neben der Automatisierung der Verifikationserstellung die Reduzierung von Inkonsistenzen und Implementierungsfehlern aufgrund einer nicht eindeutigen Spezifikation.

Die Automatisierung des Systementwurfs wie auch der Verifikationserstellung birgt das Potenzial, den Aufwand für diese Bereiche der SoC-Entwicklung erheblich zu senken. Dabei bleiben jedoch die Defizite in der Systemkonzeptentwicklung wie auch der Interpretierbarkeit der Spezifikation meist unverändert.

Abgrenzung zur Arbeit

Die in diesem Kapitel vorgestellten Lösungen für die modellgetriebene Systementwicklung und Verifikation bieten Ansätze zur modellbasierten Codegenerierung. Da sich die Lösungen dabei meist auf die Realisierung von Generatoren oder die Entwicklung eines Generierungs-Flows für einen spezifischen Anwendungsfall fokussieren, haben diese nur einen indirekten und geringen Einfluss auf die Anforderungen **A1**, **A2** und **A3,** welche verschiedene qualitative Aspekte der Spezifikation betreffen. Darüber hinaus beschäftigen sich die Arbeiten aus der Literatur nicht mit Themen wie der Integrierbarkeit der Lösungen in den bestehenden SoC-Entwicklungs-Flow (**A4**), dem Einfluss der Lösungen auf den Aufwand bei der Spezifikation (**A5**) oder der Anwenderfreundlichkeit der jeweiligen Lösungen (**A6**).

Zu Beginn des Kapitels werden Lösungen für die Automatisierung des Systementwurfs verschiedener Domänen und als Teil dessen die Erstellung von Virtuellen Prototypen mittels modellbasierter Codegenerierung diskutiert. Die Automatisierung ermöglicht in erster Linie die Reduzierung des manuellen Aufwandes für die Implementierung des Entwurfs (**A7**), hat jedoch darüber hinaus in der Regel indirekt einen positiven Einfluss auf die Konsistenz zwischen implementiertem System und Spezifikation, da Interpretationen der Spezifikation vermieden werden können.

Die Lösungsansätze zur Automatisierung der Verifikationserstellung haben in erster Linie die Reduzierung des Aufwandes in diesem Bereich (**A8**) zum Ziel, beschäftigen sich jedoch darüber hinaus mit der Formalisierung der Spezifikation und führen damit zu einer Steigerung der Eindeutigkeit. Wie bereits beschrieben, führt eine Steigerung der Eindeutigkeit der Spezifikation seinerseits zu einer Reduzierung der Wahrscheinlichkeit für Implementierungsfehler (**A2**) und Inkonsistenzen (**A3**).

3.3 Abgrenzung zur Arbeit

In dem nachfolgenden Abschnitt folgt die Abgrenzung der vorliegenden Arbeit von den in der Literatur beschriebenen Lösungen. Die hier entwickelte und vorgestellte Methode soll die in Abschnitt 1.1 definierten Anforderungen erfüllen. Daher werden die Lösungen der Literatur, wie zu Beginn des Kapitels beschrieben, auf die Erfüllung der in dieser Arbeit gestellten Anforderungen **A1–A8**

geprüft und somit eine Abgrenzung der Arbeit erreicht. Zusätzlich hebt sich die hier entwickelte Methode von allen dem Autor bekannten Ansätzen und Methoden aus der Literatur durch die folgenden Merkmale in Summe maßgeblich ab:

- Die modellbasierte Entwicklungsmethode zielt auf die Steigerung der Effizienz in der SoC-Entwicklung über alle Phasen des SoC-Entwicklungs-Flows hinweg ab. Die Methode ermöglicht hierfür einen modellbasierten Ansatz von der Analyse der Kundenforderungen über die Spezifikation des SoCs und die Entwicklung des Systemkonzeptes bis hin zur Automatisierung des Entwurfs und der Verifikationserstellung.
- Die Methode beinhaltet eine an den Bereich der SoC-Entwicklung angepasste domänenspezifische Formalisierung einer standardisierten und grafischen Modellierungssprache.
- Die Bereitstellung eines Automotive-SoC-Profils auf Basis einer „Stakeholder“-Befragung, welches den domänenspezifischen Anforderungen angepasst wurde und somit möglichst optimal die Bedürfnisse der Anwender/-innen erfüllt.
- Die Möglichkeit, die Methode in den bestehenden Entwicklungs-Flow mit dessen spezifischen Arbeitsweisen, Tools und Beschreibungsformaten zu integrieren.
- Einen kombinierten Ansatz für die Spezifikation aus Systemmodell und in natürlicher Sprache beschriebener Systemanforderungen.

Aufgrund der dargestellten Eigenschaften unterscheidet sich die Methode bereits von den in dieser Arbeit diskutierten Ansätzen aus der Literatur. So bieten reine Modellierungsmethoden eine eindeutigere Beschreibungsform bei der Systementwicklung und unterstützen dabei in der Regel eine umfangreiche und modellbasierte Analyse zu Beginn der Entwicklung. Eine ausführliche Analyse und modellierte Beschreibung des Systems kann somit benötigte nachträgliche Änderungen der vertraglich vereinbarten Anforderungen reduzieren (**A1**). Die Definition des Modells als Teil der Spezifikation und die Formalisierung der Spezifikation erhöhen darüber hinaus die Eindeutigkeit der Spezifikation. Durch diese zugewonnene Eindeutigkeit müssen die Entwickler/-innen die Informationen nicht mehr interpretieren, wodurch das Risiko für Implementierungsfehler (**A2**) und Inkonsistenzen zwischen Spezifikation und Entwurf (**A3**) reduziert werden kann. Einzelne der hier diskutierten Lösungsansätze aus der Literatur kombinieren die modellbasierte Spezifikation des Systems mit einer standardisierten Modellierungsmethode und ermöglichen somit die Erfüllung der Anforderungen **A1**, **A2** und **A3** gleichermaßen.

Neben Defiziten in der Spezifikation und Systemkonzeptentwicklung wurden Lösungen für die Automatisierung des Systementwurfs, der Entwicklung von Virtuellen Prototypen (**A7**) und der Verifikationserstellung (**A8**) aus dem Bereich der Eingebetteten Systeme diskutiert.

In der nachfolgenden Tabelle 3.1 wird sowohl der Vergleich der vorgestellten Lösungen aus der Literatur als auch die Erfüllung der genannten Anforderungen zusammengefasst dargestellt.

Tabelle 3.1 Vergleichende Einordnung der Lösungen aus der Literatur

Abschnitt	Lösungsansatz	Lösung	A1	A2	A3	A4–A6	A7	A8
Modellbasierte Systementwicklung	Modellierungsmethode	[32]	✓	✗	✗	–	–	–
		[33]	✓	✓	✗	–	–	–
		[22]	✓	✗	✗	–	–	–
		[34]	✓	✗	✗	–	–	–
	Spezifikationslogiken	[35]	✗	✗	✓	–	–	–
		[36]	✗	✗	✓	–	–	–
		[37]	✗	✗	✓	–	–	–
	Objekt–orientierte Modellierungssprachen.	[38]	✗	✓	✓	–	–	–
		[39]	✗	✓	✓	–	–	–
		[40]	✗	✓	✓	–	–	–
		[41]	✗	✓	✓	–	–	–
		[42]	✗	✓	✓	–	–	–
		[43]	✓	✓	✓	–	–	–
		[44]	✓	✓	✓	–	–	–
		[45]	✓	✓	✓	–	–	–
		[46]	✗	✓	✓	–	–	–
		[47]	✗	✓	✓	–	–	–
		[48]	✗	✓	✓	–	–	–
	SysML-basierte Ansätze	[49]	✗	✓	✓	–	–	–
		[50]	✗	✓	✓	–	–	–
		[51]	✓	✓	✓	–	–	–
		[52]	✗	✓	✓	–	–	–
		[53]	✗	✓	✓	–	–	–

(Fortsetzung)

Tabelle 3.1 (Fortsetzung)

Abschnitt	Lösungsansatz	Lösung	A1	A2	A3	A4–A6	A7	A8
Modell-getriebener Systementwurf und Verifikation	Automatisierung Systementwurf	[60]	–	✗	✗	–	✓	✗
		[61]	–	✗	✗	–	✓	✗
		[64]	–	✗	✗	–	✓	✗
		[66]	–	✗	✗	–	✓	✗
		[67]	–	✗	✗	–	✓	✗
		[15]	–	✗	✗	–	✓	✗
	Autom. Verifikationserstellung	[74]	✗	✓	✓	–	✗	✓
		[75]	✗	✓	✓	–	✗	✓

4 Methode zur abstrakten Modellierung von Automotive Systems-on-Chips

Im folgenden Kapitel wird das Konzept der in dieser Arbeit entwickelten Entwicklungsmethode zur abstrakten Modellierung von Automotive Systems-on-Chips vorgestellt. Teile der Methode sowie eine Zusammenfassung der Entwicklungsmethode wurden bereits in den Arbeiten [76], [77], [78] und [79] veröffentlicht. Das übergeordnete Ziel der Methode ist hierbei, wie in Kapitel 1 beschrieben, die Steigerung der Effizienz über alle Phasen der SoC-Entwicklung hinweg.

Die Phasen der SoC-Entwicklung und die darin verwendeten Spezifikationsdokumente wurden in Abschnitt 2.3 behandelt und in Abbildung 2.5 dargestellt. Um all diese Phasen mit der modellbasierten Entwicklungsmethode dieser Arbeit abzudecken, setzt sich diese sowohl aus Anteilen zusammen, welche in erster Linie die Analyse der Kundenanforderungen und auch die Konzeptphase fokussieren, als auch aus Anteilen, welche mehrheitlich die Entwurfsphase fokussieren.

Abbildung 4.1 zeigt, aufbauend auf Abbildung 2.5, den durch Anwendung der hier vorgestellten Entwicklungsmethode angepassten SoC-Entwicklungs-Flow mit Fokus auf die darin verwendeten Spezifikationsdokumente. Der Entwicklungs-Flow wird dabei um die in der Abbildung 4.1 gelb hinterlegten Anteile erweitert. Die weiß hinterlegten Zahlen verweisen auf die nachfolgenden Abschnitte, in denen das jeweilige Element behandelt wird. Das Kapitel ist wie folgt strukturiert:

- Um gemäß Anforderung **A1** die Notwendigkeit von nachträglichen Änderungen an den zwischen Kunde und Entwickler vereinbarten Anforderungen zu verringern, wird als Teil der Entwicklungsmethode in Abschnitt 4.1 eine modellbasierte Analyse der Kundenanforderungen und des technischen Systemkontextes als Teil des Systemkonzeptes definiert.

A. Kirchner, *Entwicklung von Methoden zur abstrakten Modellierung von Automotive Systems-on-Chips*, https://doi.org/10.1007/978-3-658-38437-1_4

- Die Reduzierung von Implementierungsfehlern aufgrund einer unzureichenden Spezifikationsgenauigkeit (Anforderung **A2)** sowie die Reduzierung von Inkonsistenzen zwischen Spezifikation und Implementierung (Anforderung **A3**) durch den Ansatz der modellbasierten Systemkonzeptentwicklung, wird ebenfalls in Abschnitt 4.1 diskutiert.
- Zum Abschluss behandelt Abschnitt 4.1 Möglichkeiten zur Verbesserung der Anwenderfreundlichkeit (Anforderung **A6**), der Reduzierung des Modellierungsaufwandes selbst (Anforderung **A5**) sowie die Integration der hier entwickelten modellbasierten Methode in den bestehenden Entwicklungs-Flow (Anforderung **A4**).
- Abschnitt 4.2 beschreibt den Übergang zwischen Konzept- und Entwurfsphase und damit den in Abbildung 4.1 als Verfeinerung dargestellten Schritt der Modellierung des Verhaltens sowie der detaillierten Spezifikation der Architektur im Systemmodell. Dies stellt die Basis für die Automatisierung des Entwurfs und der Verifikation dar und ist damit essenzieller Bestandteil der Entwicklungsmethode.
- Möglichkeiten der Generierung und damit der Reduzierung des Aufwandes für die Implementierung des Entwurfs (Anforderung **A7**) sowie der Verifikation (Anforderung **A8**) werden in Abschnitt 4.3 behandelt.

Abbildung 4.2 abstrahiert den in Abbildung 4.1 dargestellten SoC-Entwicklungs-Flow und setzt das Systemmodell in den Mittelpunkt der Methode. Dabei reduziert es den Flow auf drei Blöcke:

- Die Kundenanforderungen als Teil des Lastenheftes mittels derer die Systementwicklung beginnt
- Das Systemmodell, welches stellvertretend für die angestrebte modellbasierte Entwicklungsmethode der hier vorliegenden Arbeit steht
- Der SoC, hier als grüner Block dargestellt, welcher das Ergebnis der Entwicklung in Gesamtheit repräsentiert

Als Teil der in dieser Arbeit entwickelten modellbasierten Entwicklungsmethode müssen zudem abstrahiert die folgenden zwei Übergänge ermöglicht werden:

- Schaffung einer Modellierungsmethode für die modellbasierte Spezifikation des SoCs
- Automatisierung der Implementierung des SoC-Entwurfs

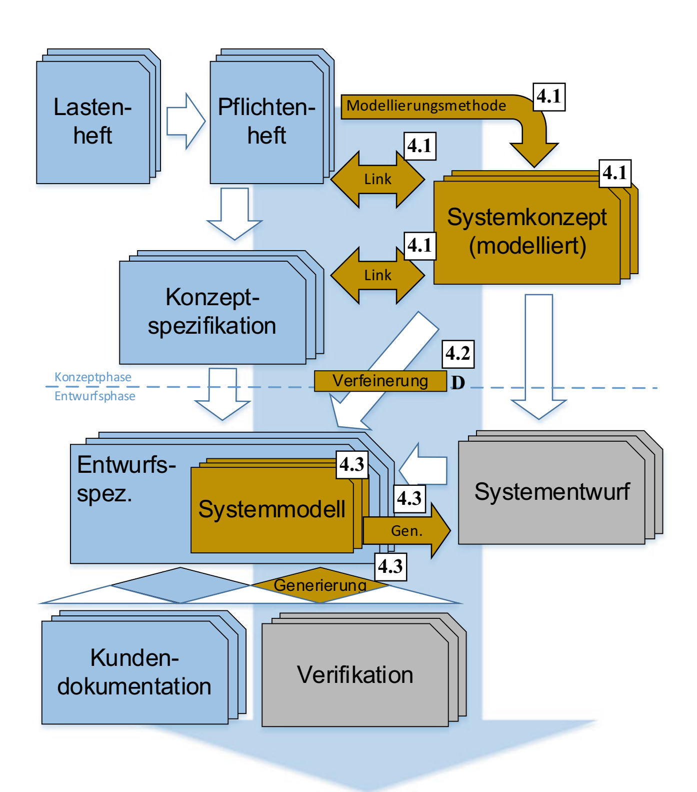


Abbildung 4.1 Erweiterter SoC-Entwicklungs-Flow

Die Abbildung 4.2 soll in den nachfolgenden Abschnitten als Grundlage dienen, um die erreichten Zwischenergebnisse zu visualisieren. Sie wird daher um den jeweils behandelten Aspekt des Abschnittes bzw. Kapitels ergänzt.

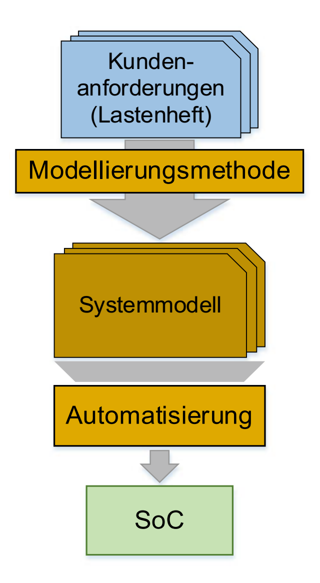


Abbildung 4.2 Systemmodellfokussierte Darstellung SoC-Entwicklungs-Flow

4.1 Modellbasierte Systementwicklung

Wie in Abschnitt 2.3 erläutert, beginnt die Systementwicklung mit der Analyse der Kundenanforderungen. Hierbei kann es aufgrund der Komplexität der SoCs und der Technologieführerschaft der Zulieferer dazu kommen, dass bereits das Schreiben des Lastenheftes in Zusammenarbeit von Kunde und Zulieferer entsteht. Doch auch wenn der Kunde bereits definierte Kundenanforderungen vorweisen kann, sollten diese analysiert und diskutiert werden, um ein gemeinsames Verständnis der beschriebenen Anforderungen zu schaffen. Dieses Vorgehen entspricht dem in Abschnitt 2.6.1 beschriebenen Top-down Ansatz für die Systementwicklung, welcher bereits seit Langem in der Literatur und der Industrie bekannt ist. Dennoch findet der Top-down-Ansatz für die Entwicklung komplexer Systeme nur selten Anwendung. Ein Grund hierfür ist der steigende Zeitdruck, welcher aus der Forderung nach kürzeren Entwicklungszeiten entsteht. Darüber hinaus bieten, wie in Kapitel 3 gezeigt, nur wenige Methoden Lösungen, um den Entwicklern/-innen eine Arbeitsweise nach dem Top-down-Ansatz nicht nur zu ermöglichen, sondern ihn dabei aktiv zu unterstützen und die Umsetzung zu fördern.

Für die Beschreibung der Kundenanforderungen im Lastenheft wie auch aller Anforderungen in den Spezifikationsdokumenten der Konzeptphase werden aktuell mehrheitlich Methoden auf Basis natürlicher Sprache für die Spezifikation

verwendet. Diese werden mit Tabellen und gezeichneten Abbildungen ergänzt. Die Beschreibung der Kundenanforderungen in natürlicher Sprache birgt fehlende Eindeutigkeit und damit Interpretationsspielraum. So kann es dazu kommen, dass trotz ausführlicher Abstimmung zwischen Kunde und Entwickler die Anforderungen unterschiedlich interpretiert werden und es im Zuge dessen zu einer Fehlkommunikation kommen kann.





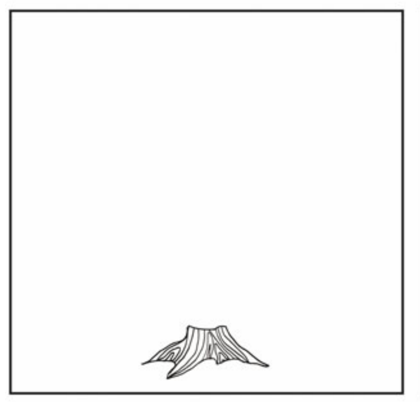






Abbildung 4.3 Schaubild „tree swing“ Systementwicklung [80] [81]

Die nachfolgende Abbildung 4.3 zeigt beispielhaft, welche Auswirkungen Defizite in der Eindeutigkeit der Spezifikation an unterschiedlichen Stellen der Systementwicklung hervorrufen können. Dabei konzentriert sich die Abbildung 4.3 auf die Rollen des Kunden und des Entwicklers als Vertreter des Zulieferers.

Die Abbildung 4.3 stellt die Problematik heutiger, vorwiegend in natürlicher Sprache verfasster Dokumentationen überspitzt dar. Der linke obere Quadrant zeigt, wie der Kunde das System mittels Kundenanforderungen im Lastenheft beschrieben hat. Im rechten oberen Quadranten ist zu sehen, wie der Entwickler anschließend das System entworfen hat. Verdeutlicht wird die Problematik der fehlenden Eindeutigkeit der Spezifikation, welche zu unterschiedlichen Interpretationen führen kann. Zusätzlich fällt die damit verbundene Fehlkommunikation nur selten oder sehr spät auf, da die Partner zwar Informationen austauschen und besprechen, jedoch die ausgetauschten Informationen auf Grundlage ihres Systemverständnisses interpretieren und verstehen. Der linke untere Quadrant weist auf ein weiteres Defizit in der Systementwicklung hin. Bei immer komplexer werdenden Systemen und bei steigendem Zeitdruck leidet die Vollständigkeit der Spezifikation. Dies führt zu Inkonsistenzen zwischen implementiertem Entwurf und Spezifikation. Ein Grund hierfür ist, dass sich Beschreibungen in natürlicher Sprache nur schwer validieren und auf Vollständigkeit prüfen lassen. Der letzte Quadrant unten rechts zeigt das System, welches eigentlich vom Kunden gewünscht und erdacht wurde. Vergleicht man nun den ersten und letzten Quadranten, wird auch hier deutlich: Selbst für den Kunden ist es schwierig, durch Verwendung heutiger Methoden und Sprachen Systeme eindeutig zu beschreiben.

Es besteht daher die Gefahr, dass bereits in dieser frühen Phase der SoC-Entwicklung Fehler in der Spezifikation entstehen, welche erst bei der Realisierung des SoCs entdeckt werden und somit zu aufwendigen und kostenintensiven Nachbesserungen führen. Die Häufigkeit solcher Nachbesserungen zu verringern, ist, wie in Abschnitt 1.1 durch die Anforderung **A1** definiert, ein Ziel der hier entwickelten Modellierungsmethode. Um dies zu erreichen, kombiniert die vorliegende Arbeit drei grundlegende Ansätze der Systementwicklung zu einer Gesamtmethode:

- **Modellbasierte Systementwicklung**
- **Entwicklung nach dem Top-down-Ansatz**
- **Formalisierung der Spezifikation**

4.1.1 Modellierungsmethode

Zur Definition einer Methode für die modellbasierte Systementwicklung bedarf es nicht in jedem Fall der Festlegung einer zu verwendenden Modellierungssprache, wie am Beispiel der in Kapitel 3 diskutierten MDA-Methode zu sehen ist. Da jedoch in dieser Arbeit eine Formalisierung der modellbasierten Spezifikation erfolgen soll und damit eine Zuweisung von Modellelementen und Diagrammen zu einer SoC-spezifischen formalisierten Bedeutung, ist dies nur durch die vorherige Auswahl einer Modellierungssprache möglich.

Ein Automotive SoC besteht, wie in Abschnitt 2.6.1 beschrieben, aus Komponenten verschiedener Domänen. Somit muss sich auch die gewählte Modellierungssprache für die Modellierung aller dieser Domänen eignen. Darüber hinaus sollten die Eindeutigkeit und die Lesbarkeit der Informationen erhöht werden. Dies wird durch die Verwendung einer standardisierten und grafischen Modellierungssprache begünstigt. Durch die bereits beschriebene modulare Implementierung der SoCs eignet sich besonders die Verwendung einer objektorientierten Modellierungssprache. Diese Anforderungen an die Modellierungssprache werden durch die in Abschnitt 2.5.2 vorgestellte SysML vollständig erfüllt. Daher baut die nachfolgend konzeptuell behandelte Methode dieser Arbeit auf der SysML als Modellierungssprache auf.

Um neben der Auswahl der Modellierungssprache die Kombination der zuvor genannten Ansätze für die Systementwicklung zu ermöglichen, ist die Definition einer an die speziellen Anforderungen der SoC-Entwicklung angepassten Modellierungsmethode notwendig. Dabei dient die in dieser Arbeit beschriebene Methode nicht nur zur Definition einer Arbeitsweise für die modellbasierte Systementwicklung von SoCs. Als Teil der Methode wird zudem eine Modellstruktur definiert, welche die Anwender/-innen an diese Arbeitsweise weitestgehend binden soll. Die Modellierungsmethode implementiert dabei eine Arbeitsweise nach dem Top-down-Ansatz und baut zudem auf der in Kapitel 3 vorgestellten SPES-Methode von [34] auf. Durch die Definition der Modellstruktur als Teil der Methode wird nicht nur ein Wiedererkennungswert zwischen den Modellen und damit eine allgemeine Lesbarkeit aller Modelle erreicht, es wird zudem der Modellierende in gewissem Maße an die Verwendung der Methode gebunden. Nachfolgend soll die SysML-basierte Modellierungsmethode anhand der in Abbildung 4.4 gezeigten Modellstruktur dieser Arbeit beschrieben werden.

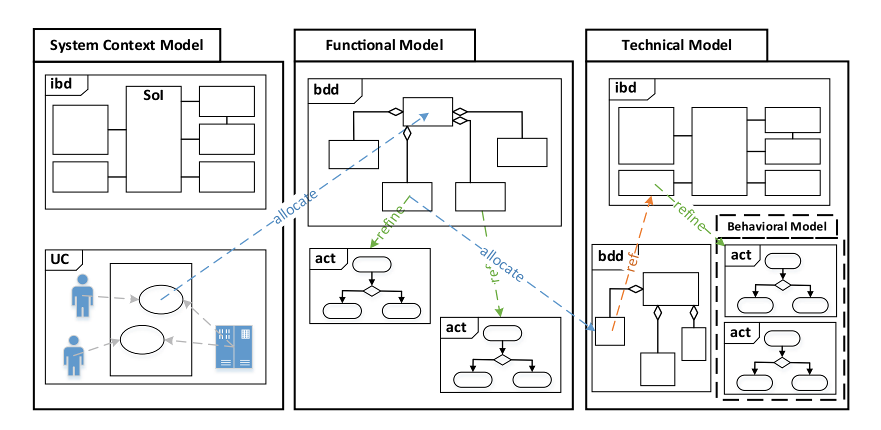


Abbildung 4.4 Modellstruktur

Die Struktur ist dabei in drei Submodelle unterteilt, deren Inhalt sich an dem in Abschnitt 2.6.1 beschriebenen analytischen Vorgehen des Top-down-Ansatzes orientiert, und weist zudem den Diagrammen der SysML eine formale Bedeutung für die SoC-Entwicklung zu. Damit wird nachfolgend der Grundstein für die Formalisierung der Spezifikation gelegt. Die Aufteilung des Modells in Submodelle erfolgt durch das in Abschnitt 2.5 vorgestellte Modellelement, genannt Package.

System Context Model

Die Entwicklung des Systemkonzeptes, welchem das Modell in der Konzeptphase entspricht, beginnt, wie in Abschnitt 2.3 beschrieben, mit der Analyse der Kundenanforderungen. Diese erfolgt traditionell als Anwendungsfallanalyse mittels Use Case Diagram und ist Teil des *System Context Model*. Dabei muss geklärt werden, welche Anwender/-innen in welchem Anwendungsfall mit dem zu entwickelnden System interagieren. Um diese Analyse bei der Entwicklung komplexer Systeme zuverlässig durchführen zu können, ist oftmals die Zusammenarbeit zwischen Kunde und Zulieferer als Technologieführer nötig. Hierzu bietet der in dieser Methode definierte modellbasierte Ansatz durch die grafische Repräsentation der Informationen eine ideale Grundlage für die erfolgreiche Kommunikation zwischen allen Beteiligten. Darüber hinaus findet, anders als im bisherigen Vorgehen, die Modellierung der Anwendungsfälle durch den modellbasierten Ansatz direkt als Teil des Systemkonzeptes statt und ist somit später Teil der Spezifikation. Dies ermöglicht es allen beteiligten Entwicklern/-innen, zu einem späteren Zeitpunkt den Ursprung der daraus entstehenden Systemanforderung nachzuvollziehen.

In Abschnitt 2.4 wurde, als Teil der Fallstudie von [19], das fehlende Wissen über den Systemkontext des zu entwickelten Systems als Defizit von den Ingenieuren/-innen aus dem Bereich der Eingebetteten Systeme angegeben. Um dieses Defizit zu minimieren oder im Idealfall zu beheben, soll in der hier vorgestellten Methode, als Teil des *System Context Model,* eine Analyse und Modellierung des technischen Systemkontextes erfolgen. Diese Informationen sind besonders im Bereich der Eingebetteten Systeme und damit der SoC-Entwicklung wichtig, da diese stets Teil eines übergeordneten Systems sind und somit eine Vielzahl an technischen Schnittstellen und Funktionalitäten als Anforderungen aus den umliegenden Systemen resultiert. Für die Modellierung des Systemkontextes wird das in der SysML enthaltene Internal Block Diagramm (IBD), welches in Abschnitt 2.5.2 bereits grundlegend besprochen wurde, verwendet.

Functional Model

Es folgt der Übergang zum *Functional Model* und damit die Ableitung von geforderten Funktionen aus den Anwendungsfällen, welcher der SoC erfüllen muss. Dieser Arbeitsschritt ist in Abbildung 4.5 als blauer *allocate*-Pfeil beispielhaft dargestellt.

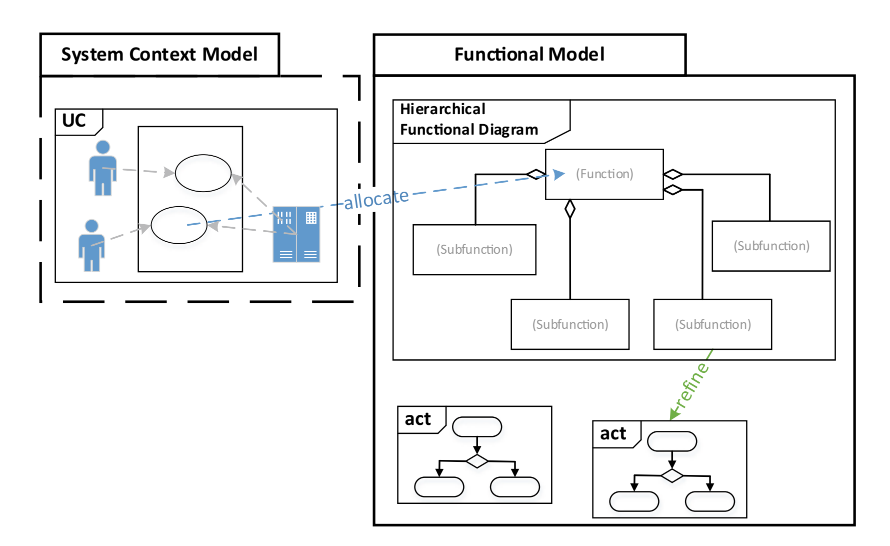


Abbildung 4.5 Auszug Functional Model Modellstruktur

Dabei entsprechen die extern sichtbaren Funktionen des SoCs den Anwendungsfällen und definieren, welche Funktionalität den Anwendern/-innen durch den SoC

als Gesamtsystem zur Verfügung gestellt wird. Die so erhaltenen Funktionen entsprechen der obersten Ebene im Block Definition Diagram (BDD) des *Functional Model.* Im nächsten Schritt werden die Funktionen entsprechend dem Top-down-Ansatz in Teilfunktionen dekomponiert. Die Dekomposition entspricht dabei der Analyse, welche interne Funktionen der SoC benötigt, um die externen vom Kunden geforderten Funktionalitäten zu gewährleisten. Das *Functional Model* soll keine Informationen enthalten, wie die Funktionen technisch umgesetzt werden, sondern lediglich eine lösungslosgelöste Analyse der benötigten Funktionen ermöglichen. Durch dieses Vorgehen wird zum einen die Wahrscheinlichkeit reduziert, dass benötigte Funktionen nicht berücksichtig werden. Zum anderen senkt es die Gefahr, dass Funktionen realisiert werden, welche nicht vom Kunden gefordert wurden oder zur Erfüllung dieser dienen. Das BDD des *Functional Model* wird nachfolgend als *Hierarchical Functional Diagram* bezeichnet.

Zusätzlich zur reinen Dekomposition der Funktionen werden im *Functional Model* die Teilfunktionen mittels verhaltensbeschreibender Diagramme logisch verknüpft. Das Activity Diagram steht in Abbildung 4.5 stellvertretend für die verhaltensbeschreibenden Diagramme. Darüber hinaus stehen den Anwendern/-innen das Sequence Diagram und das State Machine Diagram zur Verfügung. Die Beziehung zwischen den Teilfunktionen des BDDs als Verfeinerungsbeziehung zum Activity Diagram wurde in Abbildung 4.5 als grüner *refine*-Pfeil dargestellt. Die Verfeinerungsbeziehung steht für die Modellierung der nächsttieferen Abstraktionsebene eines Modellelements.

Technical Model

Beim Übergang zum *Technical Model* erfolgt die Zuordnung der Teilfunktionen aus dem *Hierarchical Functional Diagram* zu funktionalen Modulen. Zum Zeitpunkt der Konzeptphase finden diese Entscheidungen rein konzeptuell statt. Die Modellierung des *Technical Model* beginnt wie beim *Functional Model* mit der Modellierung des BDD als *Hierarchical Technical Diagram.* Dabei werden die zur Realisierung des SoCs benötigten Module analysiert und modelliert. Gleichzeitig wird die Zuordnung der benötigten Teilfunktionen zu den jeweiligen Modulen als *allocate*-Beziehung, wie in Abbildung 4.4 und Abbildung 4.6 dargestellt, modelliert. Im IBD des *Technical Model* wird anschließend die Architektur des SoCs als Ganzes sowie einzelner Teilkomponenten modelliert – im Nachfolgenden als *Architecture Diagram* bezeichnet. Dabei werden die einzelnen Module spezifiziert und die Anforderungen an diese in der Konzeptspezifikation dokumentiert.

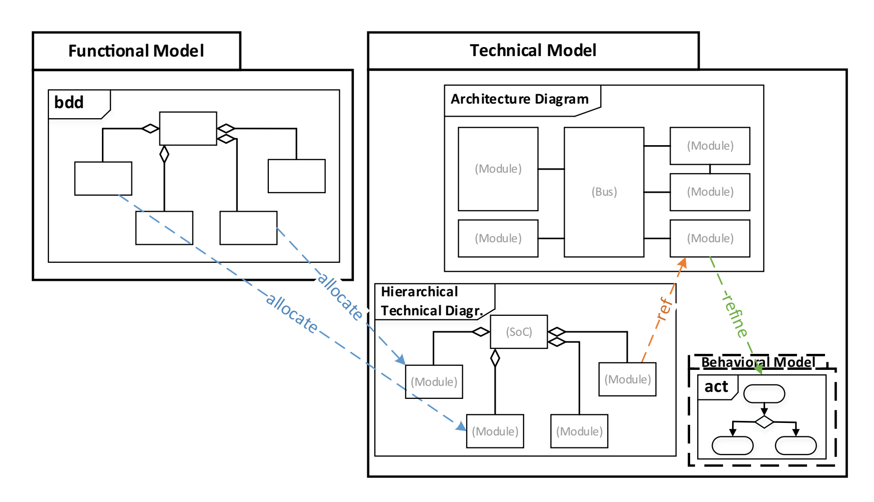


Abbildung 4.6 Auszug Technical Model Modellstruktur

Durch die beschriebene Modellierungsmethode und die damit verbundene Modellstruktur wird eine Nachverfolgbarkeit der Anforderungen von den Modulanforderungen bis hoch zu den ursprünglichen Kundenanforderungen ermöglicht. Somit kann das in Abschnitt 2.4 (Abbildung 2.10) identifizierte Defizit der nicht ausreichend vorhandenen „Traceability“ (Nachverfolgbarkeit) durch die hier vorgestellte Modellierungsmethode minimiert und somit die Nachverfolgbarkeit gesteigert werden.

Das *Technical Model* enthält in der definierten Struktur ein weiteres Submodell, das *Behavioral Model*. In diesem werden verhaltensbeschreibende Diagramme verwendet, um die Funktionalität einzelner Module und Komponenten auf technischer Ebene zu beschreiben. Diese technische und implementierungsnahe Beschreibung soll nachfolgend als Verhalten bezeichnet werden. Sie wird in der Regel erst in der Entwurfsphase Teil des Modells. Daher folgt eine nähere Beschreibung im nachfolgenden Abschnitt 4.2.1.

Die hier definierte Modellstruktur wird mittels Packages im SysML-Modell realisiert und soll für alle SoC-Projekte gleichermaßen gelten. Um den Aufwand bei der Modellierung zu reduzieren und darüber hinaus die Anwender/-innen an die Verwendung der Modellstruktur zu binden, wird diese den Anwendern/-innen als ein Template zur Verfügung gestellt. Durch dieses Vorgehen wird ein erster

Schritt in Richtung formalisierter Spezifikation erreicht und zudem die Entwickler/-innen durch das Arbeiten in der Modellstruktur in einer Arbeitsweise nach dem Top-down-Ansatz unterstützt.

Zur Verfügung gestellt wird die in dieser Arbeit definierte Struktur des Modells, fertig modelliert als Teil des sogenannten Automotive-SoC-Profils, welches ebenfalls im Zuge der hier vorliegenden Arbeit entwickelt wurde. So wie die SysML als Profil der UML darstellbar ist, so lassen sich auch innerhalb der SysML domainspezifische Profile erstellen. Dabei können Profile, vergleichbar mit einem Package, abhängig vom Anwendungsfall verschiedenste Diagramme und Modellelemente enthalten [25]. Somit lässt sich ebenso die Modellstruktur, welche durch Packages modelliert wird, als Teil des Automotive-SoC-Profils zur Verfügung stellen. Der gesamte Inhalt des Automotive-SoC-Profils ist als eine Art Werkzeugkasten zu verstehen. Er wurde im Zuge der hier vorliegenden Arbeit in Zusammenarbeit mit SoC-Entwicklern/-innen definiert. Auf das Automotive-SoC-Profil in seinem gesamten Umfang wird in Abschnitt 4.1.3 näher eingegangen. Da das Automotive-SoC-Profil einen direkten Einfluss auf die Themen Formalisierung, Anwendbarkeit, Fehleranfälligkeit und Aufwand für die Modellierung des Systemkonzeptes hat, wird es als Teil der hier vorgestellten modellbasierten Entwicklungsmethode behandelt.

Durch die SysML-basierte Modellierungsmethode für die SoC-Entwicklung, gepaart mit der definierten und als Template gelieferten Modellstruktur, wurden die Ansätze **modellbasierte Systementwicklung** und **Top-down**-orientierte Arbeitsweise in die hier entwickelte Entwicklungsmethode integriert. Ebenso erfüllt die Definition einer festen Modellstruktur einen ersten Schritt in Richtung **formalisierte Spezifikation.** Um jedoch eine nahezu vollständige Eindeutigkeit der modellierten Spezifikation über alle in Abschnitt 2.3 beschriebenen Spezifikationsdokumente hinweg zu erreichen, bedarf es der domänenspezifischen Formalisierung der SysML für die Automotive-SoC-Entwicklung. Dies bedeutet eine feste Zuweisung von Modellelementen und Diagrammen zu einer SoC-spezifischen eindeutigen Interpretation.

Die SysML hat das Ziel, den Systementwicklern/-innen übergreifend über eine Vielzahl von Domänen hinweg eine Sprache zur modellbasierten Beschreibung zu bieten. Stellt man hierbei einen Vergleich zwischen grafischen Modellierungssprachen und natürlichen Sprachen her, entsprechen die Modellelemente in gewisser Weise den Vokabeln einer natürlichen Sprache und die Diagramme einer Art Grammatik. Dabei weist die Grammatik der SysML wie die Grammatik natürlicher Sprachen gewisse Freiheitsgrade für die Anwender/-innen auf. So ist es auch in der SysML nicht definiert, wie die Diagramme zu verwenden sind und welche Bedeutung bestimmte Elemente bezogen auf ein System besitzen. Durch

diese Freiheitsgrade erhält man die hier ursprünglich gewünschte Generalisierung der Sprache. Im Kontext einer Systembeschreibung in Form beispielsweise eines Systemkonzeptes ist es jedoch essenziell, das System eindeutig beschreiben zu können und den Interpretationsspielraum so gering wie möglich zu halten. Denn ist die Spezifikation des Systems nicht eindeutig beschrieben, kann es zu einem unterschiedlichen Verständnis des Systems zwischen Kunde und Entwickler oder zwischen den Entwicklern/-innen kommen. Dies führt immer wieder zu Fehlern in der Implementierung, die zudem oftmals erst in einer sehr späten Phase der Implementierung entdeckt werden [22].

4.1.2 Formalisierung der modellierten Spezifikation

Nachfolgend soll als Teil der hier entwickelten Methode die Formalisierung für die modellbasierte Spezifikation von SoCs aufgezeigt werden. Auf die Bedeutung der Diagramme wurde bereits als Teil der Beschreibung der Modellstruktur in Abschnitt 4.1.1 eingegangen. Nachfolgend soll die Verwendung der Diagramme näher behandelt sowie die Bedeutung der jeweiligen Modellelemente bezogen auf die Automotive-SoC-Entwicklung formalisiert werden. Darüber hinaus findet als Teil der Formalisierung eine Einschränkung der für die Methode anzuwendenden SysML-Modellelemente im jeweiligen Diagramm statt. Zur Veranschaulichung soll für die wichtigsten Diagramme exemplarisch die Zuweisung von Modellelement zur formalen Bedeutung in den nachfolgenden Tabellen – Tabelle 4.1, 4.2, 4.3, 4.4, 4.5 und Tabelle 4.6 – vorgestellt werden.

System Context Model

Das Package, genannt *System Context Model,* stellt den Startpunkt der modellbasierten Systemkonzeptentwicklung dar. Es dient der Analyse der Kundenanforderungen und auch des technischen Systemkontextes.

- Das **Use Case Diagram** dient dabei zur Analyse, welche Funktionalität des SoCs als Teil eines übergeordneten Systems vom Kunden gefordert wird. Hierbei sind die Aktoren und Nutzer in der Regel andere Systeme des übergeordneten Systems, wie zum Beispiel eine ECU, welche mit dem SoC interagiert.

Tabelle 4.1 Auszug Formalisierung Use Case Diagram

SysML-Modellelement	Formalisierte Automotive-SoC-Interpretation
Use Case	Anwendungsfälle bzw. Funktionen, welche durch andere Subsysteme des übergeordneten Systems vom SoC benötigt werden
Actor	Anderes System, welches mit dem SoC interagiert oder mit dem der SoC interagiert
Boundary Box	Repräsentation des SoCs im Diagramm
Association	Zuweisung, welches externe System für welchen Anwendungsfall mit dem SoC interagiert
Dependency	Dient der Darstellung von Beziehungen zwischen den Anwendungsfällen untereinander, z. B. Ableitungsbeziehung zwischen Anwendungsfällen

- Das **Internal Block Diagram,** als *System Context Diagram* benannt, dient der Modellierung des technischen Systemkontextes. Dies ist bei der Entwicklung von Eingebetteten Systemen und damit bei der Entwicklung von SoCs besonders wichtig, da in diesem Fall der SoC ein Teil eines übergeordneten Systems ist, welcher mit anderen Systemen erfolgreich interagieren soll. Wann der technische Systemkontext modelliert werden kann, hängt stark von dem Entwicklungsstand der umliegenden Systeme ab und wird daher in manchen Fällen erst in einer späteren Phase der Entwicklung final modelliert.

Tabelle 4.2 Auszug Formalisierung IBD System Context Model

SysML-Modellelement	Formalisierte Automotive-SoC-Interpretation
Part	Dient der Modellierung des SoCs sowie externer Komponenten z. B. Submodule des übergeordneten Systems
ProxyPort	Schnittstellen der Parts und damit z. B. des SoCs
Connector	Verbindung zwischen den Ports oder den Komponenten selbst; kann zudem einem Daten-Bus entsprechen

Als Ergänzung im *System Context Model* kann das Requirement Diagram hinzugefügt werden. Zum einen kann es verwendet werden, um abzubilden, welche Anforderungen durch welches Modul bzw. Submodul erfüllt wird, zum

anderen dient das Requirement Diagram dazu, Anforderungen untereinander in Beziehung zu setzen. So werden beispielsweise während der Konzeptphase Systemanforderungen aus den Kundenanforderungen abgeleitet. Diese Ableitung im Modell darzustellen ist wichtig, um später den Kontext und Ursprung der jeweiligen Systemanforderung nachvollziehen zu können. Da das Requirement Diagram jedoch nicht den SoC selbst modelliert, ist es nicht Teil der Formalisierung der Spezifikation.

Functional Model

Das *Functional Model* soll der Analyse und Modellierung der Funktionalität des SoCs dienen. Dabei liegt der Fokus auf der Analyse, welche internen Funktionen das SoC zur Erfüllung der vom Kunden geforderten Funktionalität benötigt. Dieser Zwischenschritt zwischen Analyse der Kundenanforderungen und Entwicklung einer Lösung soll eine weniger lösungsorientierte Arbeitsweise implementieren. Bei der Entwicklung komplexer Systeme, wie im Falle der Automotive SoCs, neigen Entwickler/-innen dazu, bereits bekannte Lösungsansätze wiederzuverwenden, ohne dabei eine genaue Analyse der Anforderungen an die Funktionalität durchzuführen. Aufgrund einer unzureichenden Analyse kann es jedoch zu einer fehlerhaften Dimensionierung des Systems kommen, sodass Funktionen realisiert werden können, welche ursprünglich nicht vom Kunden gefordert wurden. Zur Analyse und Modellierung der Funktionen des SoCs sollen als Teil der hier beschriebenen Methode folgende Diagramme für die formalisierte Spezifikation dienen:

- Das **Block Definition Diagram** (BDD) genannt *Hierarchical Functional Diagram,* welches zur Dekomposition und damit zur Analyse der Funktionen des SoCs dient. Dabei werden auf Grundlage einzelner Anwendungsfälle die Funktionen abgleitet, welcher der SoC nach außen für die Interaktion mit seinem Umfeld benötigt. Die Dekomposition dient der Analyse, welche Teilfunktionen der SoC daraufhin benötigt, um diese externen Funktionalitäten intern ermöglichen zu können. Das BDD wird für die hier entwickelte Methode im Umfang der zu verwendenden Modellelemente stark eingeschränkt.

- Das **Activity Diagram** dient als nächste Abstraktionsebene einer Funktion und beschreibt die Funktion als Summe aus Teilfunktionen, welche in einem Ablauf modelliert werden. Eine Besonderheit ist hierbei das Modellelement *Swimlane,* welches dazu verwendet wird, die Activities einzelnen Modulen zuzuordnen.

Tabelle 4.3 Auszug Formalisierung BDD Functional Model

SysML-Modellelement	Formalisierte Automotive-SoC-Interpretation
Block	Bildet die externen Funktionalitäten und auch die internen Teilfunktionen des SoCs als Block ab
Directed Composition	Stellt die Teile-Beziehung zwischen den Hierarchien dar, z. B. Teilfunktion ist Bestandteil einer definierten externen Funktion

Das Modellieren der *Swimlanes* erfolgt daher in der Regel erst zu einem späteren Zeitpunkt der Entwicklung.

Tabelle 4.4 Auszug Formalisierung Activity Diagram Functional Model

SysML-Modellelement	Formalisierte Automotive-SoC-Interpretation
Activity	Entspricht einer Funktion aus dem *Functional Hierarchical View.* Hinter jeder Action kann ein weiteres Activity Diagram als nächsttiefere Abstraktionsebene modelliert werden
Action Pin	Schnittstelle der Activities für Datenfluss zwischen den Funktionen
Object Flow	Modelliert den Datenfluss zwischen den Funktionen/Activities
Control Flow	Modelliert den Kontrollfluss. Damit lassen sich Wechsel ohne Datenfluss zwischen Activities modellieren
Decision Node	Stellt die Abfrage einer Bedingung bzw. Schleifen dar, z. B. If-Abfrage (entspricht Logikelement „Oder“)
Merge Node	Verbindet die durch die Decision Node geteilten Flows
Swimlane	Dient der Zuweisung einzelner Funktionen/Activities zu definierten Komponenten und Modulen, welche diese später bearbeiten

- Das **Sequence Diagram** dient zur Modellierung einer festgelegten Sequenz und legt dabei der Fokus auf die Modellierung der Kommunikation zwischen einzelnen Funktionen. Es kann dabei beispielsweise dazu dienen, einen möglichen Durchlauf eines Activity Diagram zu modellieren.

Technical Model

Haben Analyse und Modellierung der Funktionalität im *Functional Model* einen gewissen Reifegrad erreicht, kann damit begonnen werden, die einzelnen Funktionen und Teilfunktionen den Modulen zuzuordnen. Dies geschieht im *Technical Model.*

Das *Technical Model* ist ein entscheidender Schritt in Richtung Systementwurf, da bei der Modellierung der im *Technical Model* enthaltenen Diagramme erste Entwurfsentscheidungen getroffen werden. Diese Entscheidungen sind rein konzeptuell und erfolgen lediglich auf System-Ebene. Spätestens bei der Modellierung des *Technical Model* erfolgt die Einteilung der Funktionen in die Domänen Analog- und Digitaltechnik und es folgen erste Entscheidungen, welche Funktionen durch die Software bzw. Hardware realisiert werden. Das *Technical Model* beinhaltet dabei zwei Arten von SysML-Diagrammen. Die Modellierung ist hierbei vergleichbar mit dem *Functional Model.*

- Mittels **BDD,** genannt *Hierarchical Technical View,* wird eine Dekomposition des SoCs vorgenommen. Dabei wird der SoC auf System-Ebene in Module zerlegt. Parallel erfolgt die bereits erwähnte Allokation, welche der Module für die Bearbeitung welcher Funktion aus dem *Functional Model* zuständig sind.

Tabelle 4.5 Auszug Formalisierung BDD Technical Model

SysML-Modellelement	Formalisierte Automotive-SoC-Interpretation
Block	Bildet den SoC sowie auch die Teilkomponenten und Submodule im BDD ab
Directed Composition	Stellt die Teile-Beziehung zwischen den Hierarchien dar, z. B. Modul ist Teil des SoCs. Bei der Modellierung der Directed Composition wird ein Part mit demselben Namen erzeugt

- Das **IBD** im *Technical Model,* genannt *Architecture Diagram,* dient zur Modellierung der Hardware- und Software-Architektur. Dazu werden die Parts, welche durch die Modellierung der Directed Composition im *Hierarchical Technical View* entstanden sind, zu einer Architektur zusammengesetzt.

Tabelle 4.6 Auszug Formalisierung IBD Technical Model

SysML-Modellelement	Formalisierte Automotive-SoC-Interpretation
Part	Entspricht einem Software- bzw. Hardwaremodul in der Architektur
ProxyPort	Schnittstellen der Parts und damit eines Moduls
Connector	Verbindung zwischen den Ports oder den Modulen selbst, kann zudem einem Bus entsprechen

Die Formalisierung der Spezifikation birgt neben der gewonnenen Eindeutigkeit Vorteile durch die einfachere Wiederverwendbarkeit, den Wiedererkennungswert zwischen einzelnen SoC-Modellen und für die Automatisierung. Auf diese Vorteile wird in Abschnitt 4.3 näher eingegangen. Jedoch führt die Modellierung der Spezifikation zu Beginn der Konzeptentwicklung im Vergleich zur Spezifikation in natürlicher Sprache zu einem Mehraufwand. Zudem werden die Anwender/-innen zu Beginn der Einführung der neuen Methode vor neue Herausforderungen gestellt, weshalb es für eine erfolgreiche Einführung sehr wichtig ist, den Aufwand zu reduzieren und den Entwicklern/-innen bei der Anwendung zu unterstützen. Hierzu wird im nachfolgenden Abschnitt näher auf die Bestandteile des Automotive-SoC-Profils eingegangen.

4.1.3 Automotive-SoC-Profil

Als Teil von Abschnitt 4.1.1 wurde das entwickelte Automotive-SoC-Profil, welches die definierte Modellstruktur den Anwendern/-innen in Form von Packages zur Verfügung stellt, bereits vorgestellt. Nachfolgend werden weitere Inhalte des in dieser Arbeit entwickelten Automotive-SoC-Profils beschrieben. Tabelle 4.7 zeigt dazu einen Auszug der wichtigsten Elemente des Automotive-SoC-Profils.

Tabelle 4.7 Auszug Elemente Automotive-SoC-Profil

Gruppe Profil-Elemente	Beschreibung
Template Modellstruktur	Gesamte Modellstruktur aus Abschnitt 4.1.1 inklusive Packages und erste Diagramme

(Fortsetzung)

Tabelle 4.7 (Fortsetzung)

Gruppe Profil-Elemente	Beschreibung
Templates Module *Technical Model*	Template eines SoCs als Block und Part, bereits versehen mit typischen Stereotypen, Datentypen, Ports usw Template eines SoC-Subblocks als Block und Part, bereits versehen mit typischen Stereotypen, Datentypen, Ports usw
Templates Ports *Technical Model*	Definition von häufig benötigten Ports inklusive deren Eigenschaften
Templates externer Standardkomponenten *System Context Model*	Vorgefertigte Induktivitäten, Kapazitäten, Widerstände, Transistoren usw. als Blöcke wie auch als Parts für die Nutzung im *System Context Diagram*
Stereotypen	Definition von SoC-spezifischen Stereotypen auf Grundlage von „Stakeholder“-Befragungen
Datentypen	Definition von SoC-spezifischen Datentypen auf Grundlage von „Stakeholder“-Befragungen
Variablen	Definition von SoC-spezifischen Variablen auf Grundlage von „Stakeholder“-Befragungen

Da, wie in Abschnitt 4.1.1 beschrieben, jedes Submodell als Teil dieser Methode definierte Diagramme enthalten kann, können sowohl die Modellstruktur als auch die typischen Diagramme bereits als Teil des Profils vorgefertigt zur Verfügung gestellt werden. Darüber hinaus wurden in Zusammenarbeit mit SoC-Entwicklern/-innen eine Reihe von Modellelementen definiert, welche im Regelfall für die Modellierung eines jeden SoC-Systemkonzeptes benötigt werden. So wurde beispielsweise ein generischer SoC-Block mit zugehörigem Part sowie ein generalisierter Block einer SoC-Komponente, ebenfalls mit zugehörigem Part, modelliert. Diese Templates werden zudem bereits mit vordefinierten Ports ausgestattet, welche mit hoher Wahrscheinlichkeit später für das jeweilige Element benötigt werden. Auf Grundlage einer „Stakeholder“-Befragung, welche Aspekte der SoC-Entwicklung aus ihrer Sicht im Modell abgebildet werden müssen, wurden zudem als Teil des Profils funktionale sowie nichtfunktionale Eigenschaften in Form von Datentypen und Variablen definiert [82]. Ein Beispiel hierfür ist die Definition einer Variable samt möglichen Werten für das ASIL.

Bei ASIL handelt es sich, wie in Abschnitt 2.2 beschrieben, um ein Risikoklassifizierungssystem [13]. Die Definition des ASIL ist fester Bestandteil der SoC-Entwicklung und wird somit in allen zukünftigen Entwicklungsprojekten benötigt. Darüber hinaus hat es eine durch die Norm definierte Anzahl an Werten und bietet sich somit optimal als Bestandteil des Automotive-SoC-Profils an. Die Definition solcher Datentypen und Variablen als Teil des Profils sorgt nicht nur für eine einheitliche Namenskonvention über alle Modelle hinweg, sondern führt dazu, dass die Entwickler/-innen sich bereits bei der Modellierung mit diesen Aspekten der SoC-Entwicklung auseinandersetzen muss. Besonders bei dem hier gezeigten Beispiel, der Bestimmung des ASIL für Module des SoCs, ist es wichtig, dass dies bei der Modellierung nicht vergessen wird.

Im Zuge der Arbeit wurden zusätzlich generische Komponenten wie Induktivitäten, Kapazitäten, Transistoren, Widerstände usw. als Parts modelliert und mit Variablen versehen. Diese werden in vielen Fällen als externe Komponenten für die Beschaltung des SoCs benötigt und dienen somit zur Modellierung des technischen Systemkontextes im *System Context Model.* Zur besseren Visualisierung werden den Parts zudem die entsprechenden elektrischen Schaltsymbole zugewiesen und diese im Modell als Teil des Parts angezeigt. Alle hier beschriebenen Templates lassen sich bei Bedarf für die Modellierung aus dem Profil in das eigentliche Modell kopieren und wiederverwenden. Anschließend können die kopierten Modellelemente im Modell weiter verfeinert und an den entsprechenden Anwendungsfall angepasst werden.

Das Automotive-SoC-Profil mit den enthaltenen Templates und vordefinierten Variablen reduziert den Aufwand für die Anwender/-innen bei der Modellierung der Spezifikation, ermöglicht durch die einheitliche Namenskonvention eine zusätzliche Vereinheitlichung der Spezifikationen und senkt zudem die Fehleranfälligkeit. Da es speziell für die Anforderungen der Automotive-SoC-Entwicklung entwickelt wurde, ist es essenzieller Bestandteil der hier vorgestellten Methode zur modellbasierten Systementwicklung. Als Teil der Modellierungsmethode wird das Automotive-SoC-Profil einem Modell eines SoC-Entwicklungsprojekts lediglich als referenziertes Objekt hinzugefügt, wie in Abbildung 4.7 dargestellt.

Der Nutzer kann den Inhalt des Profils für seine Modellierung nutzen, jedoch keine direkten Änderungen an den Inhalten im Profil vornehmen. Da das Profil jedoch im Laufe der Zeit an die Bedürfnisse der Anwender/-innen angepasst werden soll, empfiehlt sich eine zentrale Verwaltung des SoC-Profils.

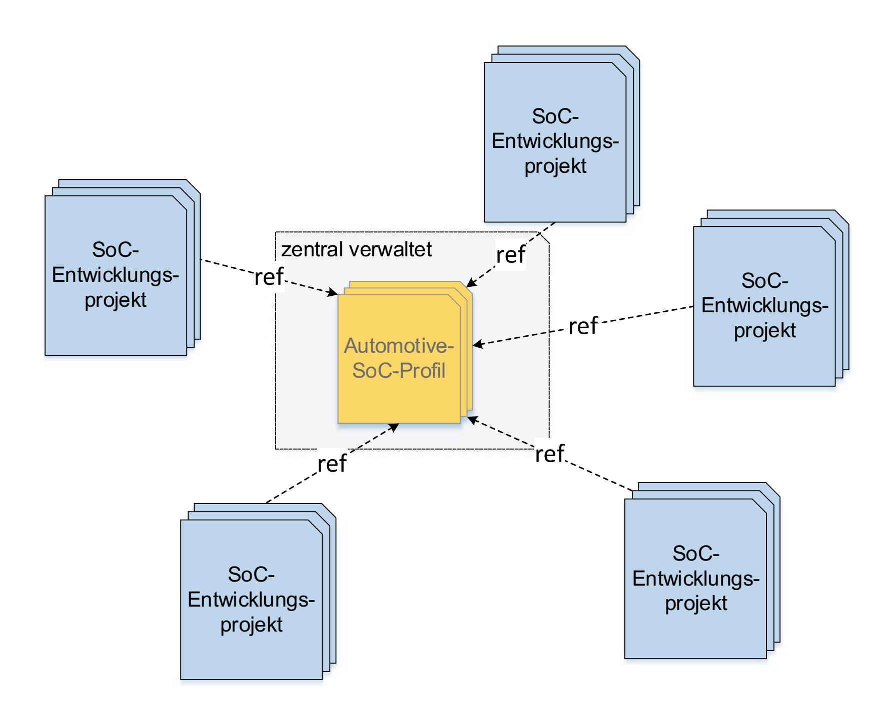


Abbildung 4.7 Verwaltung Automotive-SoC-Profil

4.1.4 Integration Modellierungsmethode in Entwicklungs-Flow

Durch die Anwendung der Modellierungsmethode auf das Systemkonzept lassen sich alle Informationen des Systemkonzeptes in nur einem zentralen Dokument spezifizieren. Verglichen mit der formlosen Sammlung aus verschiedenen Beschreibungsformen im aktuellen Flow kann so die Konsistenz gewährleistet werden. Dennoch beinhaltet das Systemkonzept Informationen, deren Spezifikation sich in anderen Formaten als der SysML besser eignen oder sich hierfür andere Formate als „Goldstandard" für bestimmte Anwendungen in der Praxis erwiesen haben. Um diese Formate mit der SysML kombinieren zu können und so die Vorteile der jeweiligen Beschreibungsform für die Spezifikation zu addieren,

werden als Teil der hier vorliegenden Arbeit Schnittstellen und Interaktionsmöglichkeiten diskutiert. Nachfolgend sollen drei der wichtigsten Formate sowie die Möglichkeiten der Kombination betrachtet werden.

Anforderungen

Das Anforderungsmanagement nimmt in dieser Arbeit eine Sonderrolle ein, da die SysML zwar die Modellierung von Anforderungen mittels Requirement Diagram bietet, diese Darstellungsform sich jedoch nur bedingt für das Anforderungsmanagement als alleinige Lösung eignet. Die Anforderungen werden in der SysML als eine Sonderform des Blocks mit ID und textueller Beschreibung dargestellt. Die Verwaltung aller Anforderungen eines SoCs als Requirements-Block im Diagramm ist jedoch nicht praktikabel, da hierdurch die Übersichtlichkeit des Diagramms stark leiden würde. Zudem werden, wie in Abschnitt 2.3 beschrieben, die Kundenanforderung in der Regel vom Kunden selbst verfasst, welcher diese meist in natürlicher Sprache spezifiziert. Diese Kundenanforderungen in Form des Lastenhefts direkt mit der darauf aufbauenden Analyse als Modell zu verlinken, ist entscheidend, um die Nachverfolgbarkeit zwischen den Anforderungen aufrechtzuerhalten.

Während der Konzeptphase entsteht zudem, wie in Abschnitt 2.3.2 behandelt, parallel zum Systemkonzept die Entwurfsspezifikation, welche nach wie vor die Systemanforderungen enthalten soll. Diese resultieren maßgeblich aus der Entwicklung des Systemkonzeptes und haben damit einen direkten Bezug zum Modell. Hierbei soll jedoch die neue Methode eine dynamische und direkte Verlinkung zwischen den Anforderungen und den Modellelementen ermöglichen. Dabei soll das Modell zum einen die Aufgaben der bisher benutzten und gezeichneten Abbildungen übernehmen, welche statisch als Bild-Format in die Spezifikationsdokumente eingefügt wurden. Zum anderen dient das SysML-Modell als Masterdokument des Systemkonzeptes und bietet somit die Möglichkeit einer formalisierten Spezifikation des Konzeptes.

Entscheidend ist hierbei, dass ein kombinierter Ansatz aus modelliertem Systemkonzept und Systemanforderungen in natürlicher Sprache geschaffen werden soll, welche gemeinsam als eine Spezifikation funktionieren. Dazu beinhaltet die hier beschriebene Methode einen toolbasierten und servergetriebenen Ansatz, welcher eine bidirektionale Verlinkung der Kunden- beziehungsweise Systemanforderungen mit den Modellelementen ermöglicht. Darüber hinaus ermöglicht diese Lösung eine Repräsentation der Elemente des jeweiligen Gegenstückes in beiden Spezifikationsdokumenten. So lassen sich zum Beispiel die in dem Anforderungsmanagement-Tool spezifizierten Systemanforderungen in das SysML-Modell übertragen und dort in den Diagrammen oder Tabellen darstellen.

Abbildung 4.8 stellt zur Veranschaulichung der Verlinkung zwischen Anforderungen und Modell einen Auszug des in Abbildung 4.1 modifizierten Entwicklungs-Flows dar.

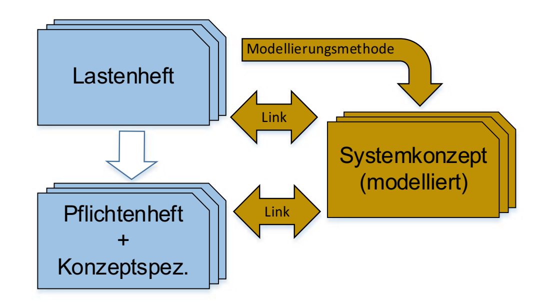


Abbildung 4.8 Kombination Anforderungsmanagement – Modell

Registerbeschreibung

Während der Konzeptphase und Konzeptentwicklung werden neben den Systemanforderungen in der Regel bereits erste Register des SoCs als Teil des Systemkonzeptes definiert. Die Registerbeschreibung beinhaltet die Definition bestimmter Speicherbereiche. Dabei werden oftmals bereits bestimmte Typen von Registern definiert und diese mit Namen versehen. Auch hier bietet die SysML kein Diagramm oder Format, in dem es möglich wäre, eine solche Beschreibung sinnvoll vorzunehmen. Des Weiteren hat sich im Bereich der Halbleiterentwickler ein XML-basiertes Austauschformat als Standard etabliert, genannt IP-XACT. IP-XACT bietet die Möglichkeit, Metadaten von Komponenten und Designs für IPs herstellerneutral zu beschreiben. Diese Beschreibungen können dann unverändert als Input für andere Werkzeuge des Entwicklungs-Flows dienen. In der hier beschriebenen Methode für die Konzeptentwicklung wird IP-XACT in erster Linie zur Beschreibung von Registern verwendet [48] [83].

Entscheidend ist hierbei die Namenskonvention, welche aus der Registerbeschreibung entsteht. So erfolgt die Ansteuerung der Register seitens Software beispielsweise unter Verwendung der in der Registerbeschreibung definierten Namen. Da sich IP-XACT als Standard in dem Bereich der Registerbeschreibung etabliert hat und diese ein essenzieller Bestandteil der Konzeptentwicklung ist, wird für die hier vorgestellte Methode ein kombinierter Ansatz gewählt. Im aktuell bestehenden Entwicklungs-Flow werden die Registerbeschreibungen manuell in Form von Tabellen in die Konzept- und später auch in die Entwurfsspezifikation übernommen oder die Namen der Register manuell in die Systemanforderungen

übertragen. Da in der Konzeptphase jedoch, wie in Abschnitt 2.3 dargelegt, lediglich konzeptuell beschrieben wird und zum Teil auf Annahmen beruht, kann es zu Änderungen in der IP-XACT-Registerbeschreibung im Laufe der Entwicklung kommen. Daraufhin müssen die Systemanforderungen überprüft und gegebenenfalls überarbeitet werden, was jedoch aus Gründen des Zeitdrucks oftmals vernachlässigt wird. Werden die Informationen aus der Registerbeschreibung nicht regelmäßig mit der Spezifikation abgeglichen, kommt es zu Inkonsistenzen und damit zur Gefahr von Implementierungsfehlern in der Entwurfsphase.

Im SysML-Modell werden die Informationen aus der Registerbeschreibung ebenfalls für die Modellierung des SoCs benötigt. So müssen beispielsweise für die Modellierung des Modulverhaltens und damit der implementierungsnahen Verhaltensmodellierung die Namen der Register zur Verfügung stehen. Die Kopplung zwischen SysML-Modell und IP-XACT führt neben der höheren Konsistenz zu einer besseren Integrierbarkeit der neuen modellbasierten Entwicklungsmethode in den aktuellen Entwicklungs-Flow. Daher wurde für die hier vorliegende Arbeit eine skriptbasierte Lösung für den Übertrag von Register- beziehungsweise Bitfeld-Namen zwischen der IP-XACT-Registerbeschreibung und dem SysML gewählt. Dazu wird eine Tabelle der Register- und Bitfeld-Namen aus IP-XACT extrahiert und diese anschließend per CSV-Import in das Modell als sogenannte *ValueProperties* importiert. *ValueProperties* sind Modellelemente der SysML, welche die Funktion einer üblichen Variablen aus dem Bereich der Programmiersprachen repräsentiert. Somit kann die Konsistenz zwischen IP-XACT-Registerbeschreibung und SysML erhöht werden, jedoch müssen mögliche nachträgliche Änderungen an der IP-XACT-Registerbeschreibung durch einen erneuten Übertrag auch im Modell aktualisiert werden. Zusätzlich zu dem reinen Übertrag der Registernamen ist es möglich, die Konfiguration des CSV-Imports anzupassen um Initialwerte, Beschreibungen der Register oder Vergleichbares ebenfalls zu übertragen.

Signalverarbeitung

Als dritte Schnittstelle zu anderen Beschreibungsformen des aktuellen Entwicklungs-Flows soll die Signalverarbeitung betrachtet werden. Zwar lassen sich mittels Verhaltens- und Strukturdiagrammen Teile der Signalverarbeitung im SysML-Modell abbilden, jedoch eignet sich die SysML in erster Linie zur Beschreibung von diskreten und ereignisgesteuerten Modellen. Für die Modellierung von Algorithmen kontinuierlicher Systeme hingegen ist diese Modellierungssprache wenig geeignet [53]. Darüber hinaus haben sich ebenfalls in der Signalverarbeitung Tools und Formate für die Beschreibung etabliert, die diese Anforderungen weitaus besser erfüllen und sich zudem als „Goldstandard" sowohl im akademischen als auch industriellen Bereich etabliert haben.

Die Signalverarbeitung ist, wie in Abschnitt 2.2 gezeigt, wichtiger Bestandteil vieler Automotive SoCs und wird in der Regel bereits abstrahiert als Teil des Systemkonzeptes spezifiziert. Dabei werden die signalverarbeitenden Komponenten des SoCs modelliert und das Verhalten der Signalverarbeitung durch Algorithmen beschrieben. Auf diese Weise erhält man eine ausführbare Spezifikation. Für die Integration der hier entwickelten Methode in den bestehenden Entwicklungs-Flow und um die jeweiligen Vorteile der beiden Beschreibungsformen für ihren Anwendungsbereich zu kombinieren, wird auch hier ein kombinierter Ansatz aus signalverarbeitenden Modellen und SysML-Modellen gewählt.

Um dies zu realisieren, wurde als Teil dieser Arbeit ein toolbasierter Lösungsansatz in die modellbasierte Methode eingebunden, welcher es ermöglicht, die signalverarbeitenden Modelle in Form von SysML-Blöcken und Parts in das SysML-Modell einzubinden. Die Blöcke und Parts der SysML werden dazu mit speziellen Stereotypen versehen und erhalten somit die Möglichkeit, ganze signalverarbeitende und zudem ausführbare Modelle hinter den SysML-Elementen zu hinterlegen. Dabei lassen sich zudem Schnittstellen für Signale zwischen den unterschiedlichen Modellen realisieren, welche die gemeinsame Simulation der Modelle ermöglichen.

Somit wird als Teil der Methode eine Möglichkeit der Modellierung geschaffen, die die Vorteile der beiden Formate kombiniert und zudem den Entwicklern/-innen die Möglichkeit bietet, weiterhin mit den etablierten Werkzeugen und Beschreibungsformen im Bereich der Signalverarbeitung zu arbeiten.

4.1.5 Zwischenergebnis

In dem vorangegangenen Abschnitt wurde ein Konzept für die modellbasierte Systemkonzeptentwicklung vorgestellt. Abbildung 4.9 erweitert den in Abbildung 4.2 abstrahierten Entwicklungs-Flow um die in diesem Kapitel vorgestellten Anteile der Methode bis zum Übergang von der Konzeptphase zur Entwurfsphase.

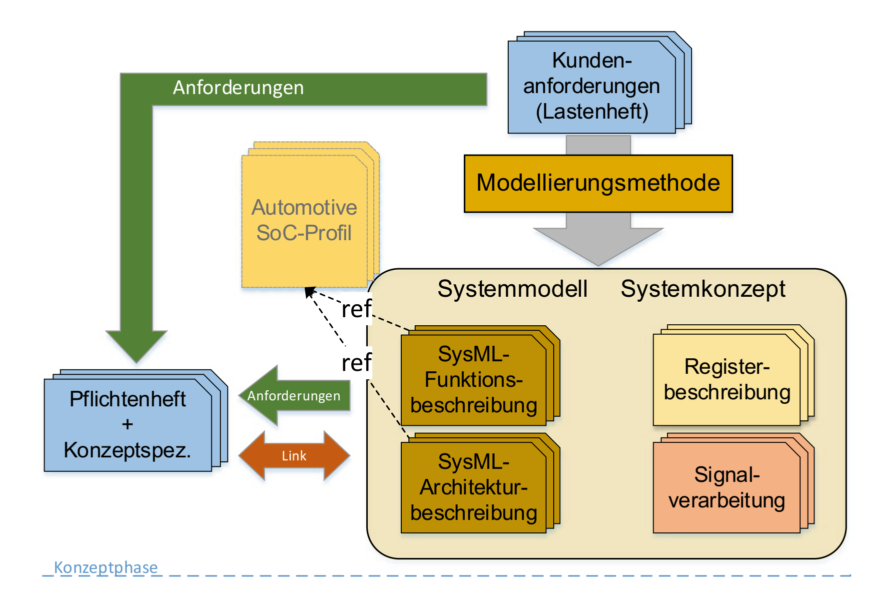


Abbildung 4.9 Zwischenergebnis modellbasierte Systemkonzeptentwicklung

In Abschnitt 4.1.1 wurde die in dieser Arbeit entwickelte Modellierungsmethode, in der Abbildung 4.9 gelb dargestellt, behandelt. Dabei unterstützt die Methode die Entwickler/-innen bei einer weniger lösungsorientierten Arbeitsweise nach dem Top-down-Ansatz und fördert so zu Beginn der Systementwicklung die ausführliche Analyse der Kundenanforderungen. Durch die ausführliche und modellbasierte Analyse der Kundenanforderungen und somit einer Analyse der Anwendungsfälle und des technischen Systemkontextes kann ein einheitliches Systemverständnis zwischen Kunde und Entwickler geschaffen werden. Hierdurch sowie durch die in Abschnitt 4.1.2 beschriebene Formalisierung wird die Wahrscheinlichkeit für unterschiedliche Interpretationen der Spezifikationsdokumente verringert und somit Folgefehler reduziert. Durch die Implementierung des Top-down-Ansatzes als Teil der Modellierungsmethode und die modellbasierte Analyse der Kundenanforderungen kann eine Senkung der Wahrscheinlichkeit für nachträgliche Änderungen an den bereits vertraglich vereinbarten Anforderungen in Lasten- und Pflichtenheft erreicht werden. Somit ermöglicht die hier vorgestellte Modellierungsmethode für die modellbasierte SoC-Systemkonzeptentwicklung die Erfüllung der in der in Abschnitt 1.1 definierten Anforderung **A1** zumindest teilweise.

Das Systemmodell entspricht in der Konzeptphase dem modellierten Systemkonzept. In Abbildung 4.9 wird das SysML-Modell abstrahiert als SysML-Funktions- und Architekturbeschreibung dargestellt und ist zu Beginn der Konzeptphase mit dem Pflichtenheft und später mit der daraus hervorgehenden Konzeptspezifikation, in der Abbildung 4.9 als roter Pfeil dargestellt, bidirektional verlinkt. Die Systemanforderungen werden aus den Kundenanforderungen abgeleitet und ergeben sich zudem aus der Entwicklung des Systemkonzepts. Die Modellierung und Formalisierung des Systemkonzeptes als ein zentrales Spezifikationsdokument in direkter Verlinkung mit den Systemanforderungen aus der Konzeptspezifikation sorgt darüber hinaus für eine Eindeutigkeit und Vollständigkeit der Spezifikationsdokumente in der Konzeptphase. Da in der später folgenden Entwurfsphase zu Beginn die Basis des Systemkonzeptes implementiert wird, ist die Steigerung der Vollständigkeit, Korrektheit und Eindeutigkeit dieses Dokuments entscheidend für eine erfolgreiche Implementierung des SoCs. Durch die Formalisierung der modellbasierten Spezifikation kann somit von einer mindestens teilweise Erfüllung der Anforderung **A2** ausgegangen werden.

Die Erweiterung der Methode mittels Automotive-SoC-Profil, in Abbildung 4.9 hellgelb dargestellt, wurde in Abschnitt 4.1.3 behandelt. Dieses sorgt durch die im Profil zur Verfügung gestellten Templates zu einer modellübergreifenden Einheitlichkeit in der Modellstruktur und Namenskonvention. Hierdurch bietet es Mechanismen zur Reduzierung des Modellierungsaufwandes und reduziert somit den Aufwand der Spezifikation, verglichen mit der aktuell angewandten Arbeitsweise. Durch das Profil werden die Anwender/-innen in der Umsetzung der Methode bei der Modellierung unterstützt und damit die Anwenderfreundlichkeit der Methode erhöht. Durch die Erweiterung der Methode um das in dieser Arbeit entwickelte Automotive-SoC-Profil wird daher ebenfalls die Erfüllung der Anforderungen **A5** und **A6** ermöglicht.

Abschnitt 4.1.4 behandelt die Kombination des mittels SysML modellierten Systemkonzeptes mit anderen etablierten Beschreibungsformen heutiger SoC-Entwicklungen. Dabei stellt die Signalverarbeitung einen essenziellen Bestandteil eines SoCs dar, für dessen Beschreibung sich andere Formate etabliert haben und somit als optimale Ergänzung dient. Ebenso hat die Schnittstelle zur Registerbeschreibung in IP-XACT einen besonderen Stellenwert, da hieraus die Namenskonvention für die Register und Bitfelder des SoCs hervorgehen, welche ebenfalls im modellierten Systemkonzept eingehalten werden müssen. Darüber hinaus bietet die hier vorgestellte Methode eine ebenfalls toolbasierte Lösung für die bidirektionale Verlinkung zwischen den Anforderungen und den Modellelementen im Systemkonzept. Die Betrachtung der Schnittstellen und die Kombination der für den jeweiligen Anwendungsfall optimalen Beschreibungsform, gepaart mit der

Möglichkeit der Verlinkung der unterschiedlichen Beschreibungsformen ermöglicht eine Steigerung der Konsistenz. Eine konsistente Spezifikation über alle Dokumente und Formate hinweg führt zu einer Minderung von Inkonsistenzen zwischen Spezifikation und Implementierung in der Entwurfsphase. Somit ermöglicht die hier beschriebene Methode die Erfüllung der in Abschnitt 1.1 beschriebenen Anforderungen **A3.** Darüber hinaus wurde durch die Entwicklung und Nutzung von Schnittstellen zwischen dem SysML-basierten Systemkonzept und bereits etablierten ergänzenden Beschreibungsformen die Integration der in dieser Arbeit entwickelten Methode in den bestehenden Entwicklungs-Flow ermöglicht und somit kann die Anforderungen **A4** als erfüllt angesehen werden.

4.2 Übergang Konzeptphase – Entwurfsphase

Wie in Abschnitt 2.3.3 beschrieben, führt der Übergang von der Konzeptphase zur Entwurfsphase zu einem Wechsel der Arbeitsweise in der SoC-Entwicklung. Dies hat einen direkten Einfluss auf die in dieser Arbeit entwickelte Gesamtmethode. Der vorangegangene Abschnitt behandelt die modellbasierte Konzeptentwicklung, hierbei entspricht das Systemmodell dem Systemkonzept und ist Basis der SoC-Entwicklung. Die Konzeptentwicklung endet in der Regel auf System-Ebene. Der genaue Zeitpunkt des Wechsels von Konzept- zur Entwurfsphase wird im Regelfall individuell durch die SoC-Architekten/-innen bestimmt.

In der Regel jedoch beginnt die Entwurfsphase mit dem Start der Implementierung des SoCs und seiner Module. Dabei werden die maßgeblich in der Konzeptphase entwickelten Modulanforderungen an die Modulentwickler/-innen übergeben. Dabei ist die Entwurfsspezifikation zentrales Spezifikationsdokument der Entwicklung und baut auf der Konzeptspezifikation auf. Die Entwicklung der Module bis runter zur Transistor-Ebene findet hierbei in entsprechenden Entwicklungstools statt.

Für die in dieser Arbeit entwickelte Methode bedeutet der Wechsel der Arbeitsweise auch einen Wechsel in der Rolle des Systemmodells. So wird das Systemmodell in der Entwurfsphase Teil der Entwurfsspezifikation.

Das Systemmodell als Teil der Entwurfsspezifikation entspricht zu Beginn der Entwurfsphase dem Systemkonzept und wird im Laufe des Entwicklungsprozesses weiterentwickelt bzw. verfeinert. Da das Systemmodell als Teil der Entwurfsspezifikation definiert ist, die Systemanforderungen und das SysML-Modell jedoch in zwei verschiedenen Tools verwaltet werden müssen, wird auch hier eine toolbasierte Lösung für die Verlinkung, wie in Abschnitt 4.1.4 beschrieben, benötigt. Dazu werden, wie bereits in der Konzeptphase, die Systemanforderungen aus der

Entwurfsspezifikation bidirektional mit den Modellelementen des Systemmodells verlinkt.

Nach dem Übergang von der Konzept- zur Entwurfsphase wird eine implementierungsnähere Modellierung des SoCs benötigt. Daher werden das Systemmodell und die Modellstruktur aus Abschnitt 4.1.1 um das *Behavioral Model* ergänzt. In Abbildung 4.10 wird das SysML-Modell abstrahiert als Architektur- und Verhaltensbeschreibung als Teil der Entwurfsspezifikation dargestellt. Nachfolgend soll der Prozess der Verfeinerung am Beispiel der Architektur- und der Verhaltensbeschreibung behandelt werden.

4.2.1 Verfeinerung (Bus-)Architektur

Bei dem Übergang von der Konzept- zur Entwurfsphase muss für die bereits modellierte Architektur des SoCs im *Technical Model* die System-Ebene als Maß der Abstrahierung und damit des Detailgrads der Modellierung stellenweise verlassen werden. Stellenweise deshalb, weil als Teil der Methode wie auch als grundlegende Eigenschaft eines Modells die Verfeinerung des Modells nicht an jeder Stelle gleichermaßen erfolgen muss. So sollen nur die Bereiche des Modells mit einem hohen Detailgrad modelliert werden, für die es sinnvoll ist. Ein Grund könnte hierbei zum Beispiel eine besonders hohe Priorität eines bestimmten Teils des SoCs sein oder aber, wie im nachfolgenden Abschnitt beschrieben, die dadurch ermöglichte Automatisierung. Die Modellierung des SoCs, besonders bei einem hohen Detailgrad, schafft, verglichen mit der Beschreibung mittels natürlicher Sprache bzw. der direkten Implementierung des Entwurfs, gefühlt einen Mehraufwand zu Beginn der Entwurfsphase. Die Automatisierung durch die Modellierung zu ermöglichen und damit die Aufwände bei der Implementierung stark zu verringern, ist daher essenziell für die hier entwickelte Gesamtmethode.

Ein Beispiel für die Verfeinerung der Architektur ist die Modellierung von On-Chip-Bussen als Teil der Hardware-Architektur. Die Busse der Hardware-Architektur werden, wie alle Teile der Architektur, als Blöcke im BDD und als Parts im IBD modelliert. Dies geschieht in der Regel bereits in der Konzeptphase. Spätestens in der Entwurfsphase reicht es jedoch nicht mehr aus, den Bus abstrahiert als nicht näher spezifizierten Part der Architektur zu modellieren. Es wird neben der Architektur der SoC-Architektur auf System-Ebene eine Modellierung der Busarchitektur benötigt. Damit wird, vergleichbar mit einer Verbindungstabelle, definiert wie die Komponenten über den Bus kommunizieren. Um dies im Modell zu realisieren, wird als nächsttiefere Abstraktionsebene hinter dem Part des Busses im *Architecture Diagram* ein weiteres IBD mit der Busstruktur

hinterlegt. Dieses IBD der Busstruktur zeigt die Module, welche über den Bus verbunden sind und löst dabei die Busarchitektur mittels *Connectoren* auf. Das bedeutet, die Parts als Repräsentation der Hardware-Komponenten werden direkt miteinander verbunden.

Dies ist nur ein Beispiel für den Prozess der Verfeinerung des Architekturmodells als Teil der Spezifikation. Vergleichbar wird bei den einzelnen Komponenten bzw. Modulen vorgegangen. So können Module in Submodule mit eigenen Architekturmodellen zerlegt werden, um die Module weiter zu spezifizieren. Darüber hinaus findet beim Übergang zwischen Konzept- und Entwurfsphase, als Teil der Modellverfeinerung, die Einteilung der in der Konzeptphase rein funktional betrachteten Module in die in Abschnitt 2.6.1 beschriebenen Domänen analoge Hardware, digitale Hardware und Software statt.

4.2.2 Verfeinerung der Verhaltensbeschreibung

Das *Behavioral Model* ist, wie in Abbildung 4.4 gezeigt, Teil des *Technical Model*. Es wird dabei allerdings einem eigenen Package zugeordnet, da hieraus später die Codegenerierung ermöglicht werden soll. Im *Behavioral Model* wird das Verhalten sowohl von der Hardware als auch von der Software des SoCs modelliert. Dabei kann zum einen das Modell als reine Spezifikation, welche das implementierte Verhalten grafisch repräsentiert und als Teil der Entwurfsspezifikation spezifiziert, dienen. Zum anderen kann die Entwicklung des Verhaltens in der Entwurfsphase ebenfalls modellbasiert erfolgen. Das bedeutet, die Entwicklung des geforderten Verhaltens der einzelnen Module beginnt im Systemmodell. Dabei wird zu Beginn das Verhalten eines Moduls noch abstrahiert modelliert und kann anschließend in mehreren Schritten und je nach Bedarf verfeinert werden. Die anfängliche Abstrahierung, welche die Modellierung bietet, kann bei der Entwicklung komplexer Funktionen und Operationen helfen und bietet zudem eine übersichtliche Kommunikationsgrundlage. Zusätzlich bietet diese Arbeitsweise einen entscheidenden Vorteil bei der Automatisierung der Entwicklung. So wird durch die formalisierte und maschinenlesbare Beschreibung mittels SysML ein Format in der Spezifikation geschaffen, welches es ermöglicht, Code auf Basis des Modells zu generieren.

Im *Behavioral Model* können alle verhaltensbeschreibenden Diagramme der SysML in einer formalisierten Form zum Einsatz kommen. Nachfolgend soll am Beispiel des Activity Diagram beziehungsweise des Flowchart Diagram die Formalisierung für das *Behavioral Model* aufgrund der hohen Relevanz für diese

Arbeit beispielhaft beschrieben werden. Dabei muss durch die doppelte Verwendung der verhaltensbeschreibenden Diagramme je nach Submodell denselben Modellelementen eine unterschiedliche formale Bedeutung zugeordnet werden.

- Das **Flowchart Diagram** als Untermenge der Modellelemente eines Activity Diagram dient im *Behavioral Model* dazu, das Verhalten der einzelnen Komponenten als Blöcke und Parts im *Technical Model* zu spezifizieren. Es wird somit dazu verwendet, die Operationen und Funktionen von Software und Hardware zu modellieren. Flowchart Diagrams bilden lediglich eine Untermenge der Modellelemente eines Activity Diagram ab, da sich hiermit keine parallelen Aktivitäten im Diagramm beschreiben lassen. Hierdurch lässt sich der im Flowchart Diagram beschriebene Ablauf leichter nachvollziehen und vereinfacht zudem die Modellierung und modellbasierte Generierung von Hardware- und Softwarefunktionen. Müssen als Teil des Modells dennoch parallele Aktivitäten modelliert werden, lassen sich diese mithilfe mehrerer Flowchart Diagrams beschreiben. Tabelle 4.8 zeigt einen Auszug der Formalisierung des Flowchart Diagram.

Tabelle 4.8 Auszug Formalisierung Flowchart Diagram Behavioral Model

SysML-Modellelement	Formalisierte Automotive-SoC-Interpretation
Activity	Entspricht einer Funktion bzw. Operation eines Moduls. Hinter jeder Activity kann ein weiteres Flowchart Diagram als nächsttiefere Abstraktionsebene modelliert werden
Action Pin	Schnittstelle der Activity für den Datenfluss zwischen den Funktionen
Object Flow	Modelliert den Datenfluss zwischen den Funktionen/Activities
Control Flow	Modelliert den Kontrollfluss. Damit lassen sich Wechsel ohne Datenfluss zwischen Activities modellieren
Decision Node	Stellt die Abfrage einer Bedingung bzw. Schleifen dar, z. B. If-Abfrage (entspricht Logikelement „Oder“)
Merge Node	Verbindet die durch die Decision Node geteilten Flows

- Das **State Machine Diagram** oder Zustandsdiagram ist weit verbreitet und findet in vielen verschiedenen Geschäftsbereichen Anwendung. Es dient dazu,

Zustände und Zustandswechsel eines Systems oder einzelner Module zu spezifizieren. Als Teil eines *State* lassen sich im Modell Actions definieren, welche beim Betreten, während der Zustand aktiv ist, oder beim Verlassen des Zustandes ausgeführt werden. Hierbei ist es möglich, einen Trigger als Auslöser für das Bearbeiten der Action zu setzen und/oder eine Bedingung zu definieren. So lässt sich ein State Machine Diagram mit beispielsweise einem Activity Diagram für die Simulation ganzer Systeme verknüpfen.
- Das **Sequence Diagram** wird im *Behavioral Model* dazu verwendet, die Kommunikation zwischen einzelnen Modulen bzw. Komponenten des SoCs abzubilden. Es kann jedoch ebenfalls verwendet werden, um eine Sequenz aus Actions unterschiedlicher Komponenten darzustellen.

4.2.3 Zwischenergebnis

Im vorangegangenen Abschnitt wurde der Übergang zwischen Konzept- und Entwurfsphase im Kontext der in dieser Arbeit vorgestellten modellbasierten Entwicklungsmethode behandelt. Dabei findet ein Wechsel der Arbeitsweise von einer rein konzeptionellen Entwicklung zu einer implementierungsfokussierten Entwicklung statt. Aufgrund dieses Wechsels in der Arbeitsweise verändert sich auch die Rolle des Systemmodells wie in Abbildung 4.10 dargestellt. So wird das Systemmodell, welches dem Systemkonzept in der Konzeptphase entspricht und als zentrales Spezifikationsdokument der Entwicklung in dieser Phase dient, zu einem Bestandteil der Entwurfsspezifikation.

Als Teil des Übergangs zwischen Konzept- und Entwurfsphase bedarf es einer Verfeinerung der Architektur sowie einer Modellierung des Verhaltens der Software- und Hardwaremodule. Dazu wurde in dem vorangegangenen Abschnitt an Beispielen aus der SoC-Entwicklung die Verfeinerung der Modellierung veranschaulicht und als Teil dessen die Formalisierung der Spezifikation für den Bereich der Verhaltensbeschreibung erweitert.

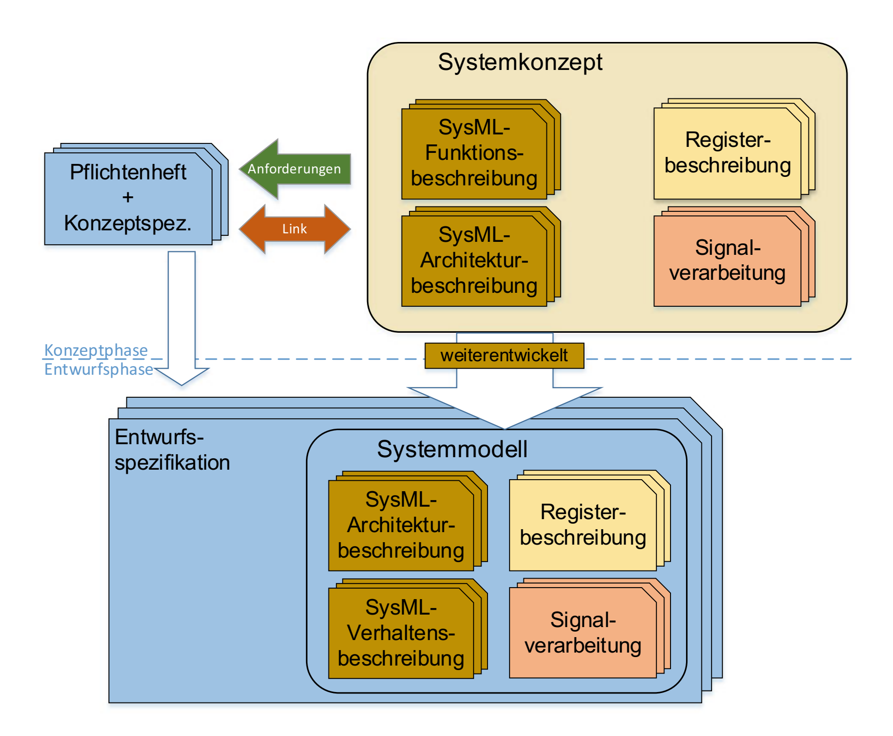


Abbildung 4.10 Übergang Konzept- zu Entwurfsphase

4.3 Modellgetriebener Systementwurf und Verifikation

Ziel der in dieser Arbeit entwickelten Methode ist es, eine Effizienzsteigerung der SoC-Entwicklung zu ermöglichen. Um dies zu erreichen, muss der Aufwand bei der Entwicklung, Implementierung und Verifikation gesenkt werden. In Abschnitt 4.1 wurde dazu eine modellbasierte Entwicklungsmethode vorgestellt, welche in erster Linie die Lesbarkeit, Eindeutigkeit und Vollständigkeit der Spezifikationsdokumente erhöht und damit die Wahrscheinlichkeit für Fehler und Missverständnisse reduziert. Somit wird indirekt der Aufwand für die Fehlerbehebung und für Nacharbeiten reduziert.

Durch die Modellierung des Systemkonzeptes, die in dieser Methode implementierte Arbeitsweise nach dem Top-down-Ansatz und die bidirektionale Verlinkung mit den Systemanforderungen entsteht jedoch darüber hinaus ein Mehraufwand, verglichen mit der Spezifikation nach der aktuellen Methode. Dieser Mehraufwand resultiert allerdings teilweise daraus, dass aktuelle Spezifikationen oftmals nur lückenhaft dokumentiert sind und ein gefühlter Mehraufwand bei der Anwendung der modellbasierten Methode durch die Behebung dieser Defizite entsteht. Die Modellierung mit SysML und die damit benötigte Nutzung entsprechender Tools stellt jedoch ungeachtet dessen besonders zu Beginn der Einführung einen Mehraufwand und damit eine große Hürde für die Entwickler/-innen dar. Daher wurde die Auswahl einer maschinenlesbaren Sprache und deren Formalisierung für die SoC-Entwicklung nicht nur zur Steigerung der Eindeutigkeit in der Spezifikation getätigt, vielmehr ermöglicht sie darüber hinaus die Automatisierung der Implementierung.

Durch die Modellierung der Spezifikation lässt sich somit als Teil der hier gezeigten Methode eine Generierung von Code verschiedenster Anwendungsbereiche des SoC-Entwurfs sowie der Verifikation ermöglichen. Als Teil des Konzeptes werden in den nachfolgenden Abschnitten die in dieser Arbeit entwickelten Methoden für die modellgetriebene Codegenerierung konzeptuell behandelt. Darauf aufbauend folgt in Kapitel 5 die Beschreibung der Implementierung für die nachfolgend konzeptuell beschriebenen Generatoren.

4.3.1 Automatisierung der Entwicklung Virtueller Prototypen

Als Möglichkeit für die Automatisierung wurde der Bereich der Entwicklung von Virtuellen Prototypen identifiziert. Wie in Abschnitt 2.6.4 beschrieben, kann durch die Verwendung von Virtuellen Prototypen die Software zu einem früheren Zeitpunkt entwickelt und getestet werden. Darüber hinaus wird durch die Verwendung von Virtuellen Prototypen ein Hardware/Software-Co-Entwurf ermöglicht. Somit können viele Fehler in einer früheren Phase der Entwicklung entdeckt werden und lassen sich dadurch leichter beheben. Hierdurch ist es möglich, Aufwand und Kosten bei der Entwicklung zu reduzieren. Jedoch entsteht bei der Erstellung eines Virtuellen Prototyps zusätzlicher Aufwand, welcher zudem parallel zu anderen Entwicklungsaufgaben anfällt. Daraus resultiert ein höherer Bedarf an Personenstunden zur selben Zeit. Um diesen Nachteil zu eliminieren, ohne dabei auf die Vorteile bei der Nutzung von Virtuellen Prototypen verzichten zu müssen, wurde ein modellbasierter Ansatz für die Generierung des Virtuellen Prototyps entwickelt.

Die Automatisierung mittels modellgetriebener Generierung bietet sich hierbei besonders an, da der Virtuelle Prototyp in der Regel abstrahiert auf System-Ebene realisiert wird. Zudem werden bei einigen Modulen lediglich die Funktionalitäten realisiert, die für die Simulation des Software/Hardware-Interfaces benötigt werden. Da im *Technical Model* die Architektur auf System-Ebene bereits modelliert ist, ist ebenso bereits ein Großteil der für die Generierung benötigten Informationen enthalten. Lediglich die implementierungsnahen Informationen der Architektur sowie die Verhaltensbeschreibung im *Behavioral Model,* welche aufgrund der anfangs noch abstrakten Modellierung des SoCs fehlen, müssen ergänzt werden. Die Verfeinerung der Modellierung ist in Abschnitt 4.2 beschrieben. Anhand der nachfolgenden Abbildung 4.11 sollen die Vorteile eines automatisierten Entwicklungs-Flows mit dem Flow aus Abbildung 2.22 verglichen werden.

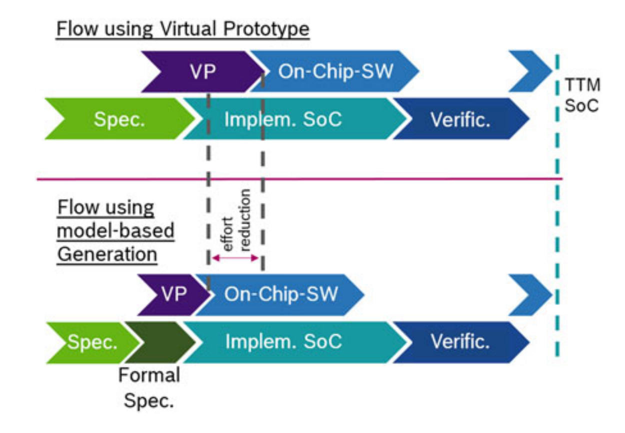


Abbildung 4.11 Automatisierte VP-Implementierungs-Flow

Die Abbildung 4.11 veranschaulicht dabei schematisch die Aufwände der einzelnen Entwicklungsphasen. Durch den modellbasierten Ansatz teilt sich die Spezifikation in zwei Abschnitte. Hinzu kommt der hier dunkelgrün dargestellte Abschnitt für die formale Spezifikation. Bei der formalen Spezifikation handelt es sich um die Modellierung des Systemmodells. Auf Grundlage dieses Modells kann der Virtuelle Prototyp generiert und damit der manuelle Aufwand bei der Entwicklung reduziert werden. Hierdurch wird nicht nur der Aufwand in Gesamtheit reduziert, es wird ebenso eine schnellere Fertigstellung des Virtuellen Prototyps und damit ein früherer Beginn der On-Chip-Softwareentwicklung ermöglicht.

Wie bereits ausgeführt, ist die Anwendung von Virtuellen Prototypen weitestgehend etabliert. Darüber hinaus wurden bereits erste Möglichkeiten der Automatisierung entwickelt, wie in Abschnitt 3.2 dargelegt. Die Arbeit von [15] beschreibt dazu einen teilautomatisierten Entwurfs-Flow für die Implementierung von Virtuellen Prototypen. Dabei konzentriert sich der in [15] beschriebene Generierungs-Flow auf die Generierung der signalverarbeitenden Anteile des SoCs auf Basis von MATLAB-Simulink-Modellen. Dieser teilautomatisierte Entwurfs-Flow für die Implementierung von Virtuellen Prototypen wurde durch die in dieser Arbeit entwickelten Lösungen zur modellbasierten Generierung der kontrollflussorientierten Anteile sowie der Architektur des SoCs erweitert. Abbildung 4.12 zeigt, als Auszug von Abbildung 3.1, noch einmal *Pfad 2* und *Pfad 3* und stellt damit den Stand der Technik zum Startzeitpunkt dieser Arbeit dar.

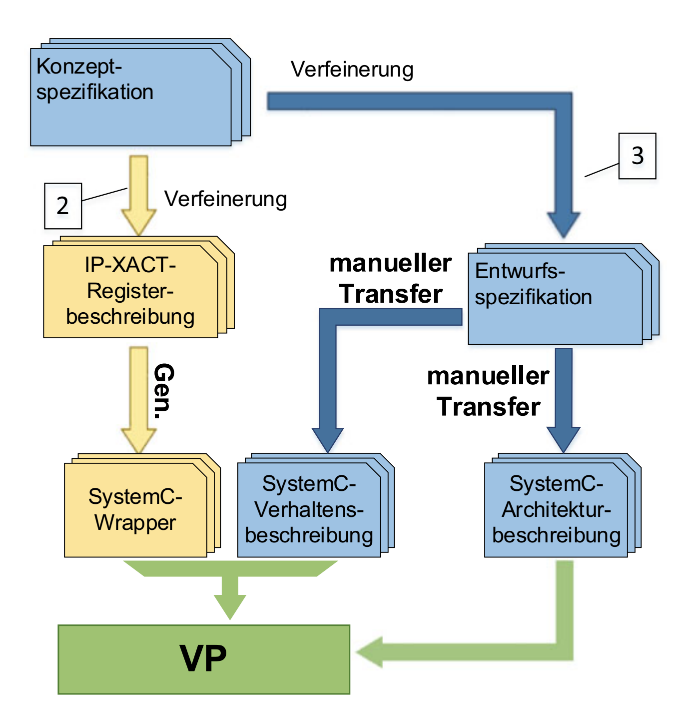


Abbildung 4.12 Auszug teilautomatisierter Entwurfs-Flow für Virtuelle Prototypen

Es handelt sich, wie in Abschnitt 3.2 beschrieben, um einen teilautomatisierten Entwurfs-Flow, der besonders in *Pfad 3* der kontrollflussorientierten Module Lücken bei der Automatisierung aufweist. Hinzu kommt, dass das in der Konzeptphase entwickelte Systemkonzept im aktuellen Entwicklungs-Flow nicht Teil der offiziellen Spezifikation ist und die hier enthaltenen Informationen manuell in die Konzeptspezifikation bzw. die Entwurfsspezifikation übertragen werden müssen. Neben dem in Abbildung 4.11 schematisch dargestellten Mehraufwand bei der manuellen Implementierung eines Virtuellen Prototyps birgt jede manuelle Übertragung von Informationen das Potenzial für Fehler und Inkonsistenzen.

Darüber hinaus müssen bereits in der Entwurfsspezifikation beschriebene Informationen manuell in Code übersetzt werden, da das ursprüngliche Format nicht maschinenlesbar ist. An eben diesen Stellen setzt die in dieser Arbeit entwickelte modellgetriebene Methode an. Da das Systemkonzept, wie in Abschnitt 4.1 beschrieben, bereits als SysML-Modell entwickelt wurde, steht es nun in einem maschinenlesbaren Format zur Verfügung. Darüber hinaus gilt es durch die Formalisierung als Teil der Spezifikation, wodurch die dort modellierten Informationen nicht manuell in ein anderes beschreibendes Format übertragen werden müssen. Das Systemkonzept zu Beginn der Entwurfsphase entspricht dem Systemmodell und muss lediglich, wie in Abschnitt 4.2 beschrieben, für die Generierung verfeinert werden, wie in Abbildung 4.13 dargestellt.

Das Systemmodell wird in der Abbildung 4.13 abstrahiert als SysML-Verhaltens- und Architekturbeschreibung repräsentiert und ist Teil der Entwurfsspezifikation. Der gestrichelte hellgelbe Pfeil zwischen IP-XACT-Registerbeschreibung und Entwurfsspezifikation veranschaulicht die in Abschnitt 4.1.4 beschriebene Schnittstelle und die damit verbundene Bekanntmachung der Registernamen im Systemmodell.

Wie in Abbildung 4.13 gezeigt, wurde im ersten Schritt die Generierung der Verhaltensbeschreibung für die kontrollflussorientierten Module realisiert. Dazu wurde das Systemmodell um die Verhaltensbeschreibung in Form des in Abschnitt 4.2.2 vorgestellten *Behavioral Model* erweitert. Zur Generierung der SystemC-Verhaltensbeschreibung wurde ein C++-Generator für die Modellierung von SystemC-Code auf Basis von SysML-Modellen konfiguriert. Zur Beschreibung des Verhaltens wurde in dieser Arbeit auf die Modellierung mittels Flowchart Diagram zurückgegriffen. Die Modellierbarkeit von Parallelität der Activity Diagrams wird im Regelfall bei der Modellierung von Funktionskörpern nicht benötigt, würde jedoch zu einer deutlich höheren Komplexität bei der Generierung führen. Die Informationen über die Architektur werden zu diesem Zeitpunkt, wie in Abbildung 4.13 gezeigt, weiterhin manuell übertragen.

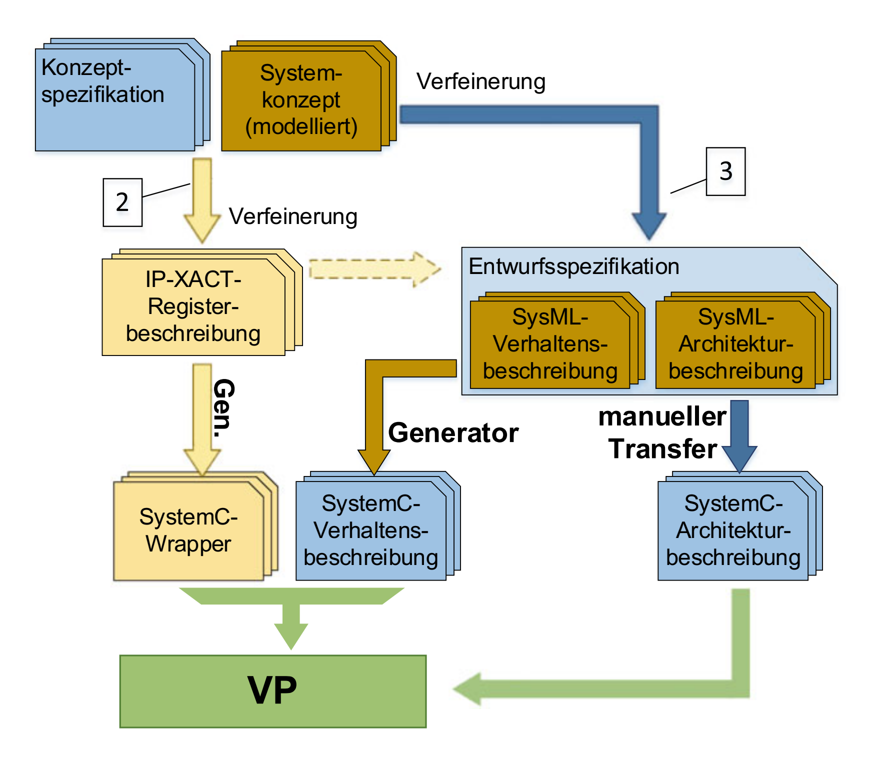


Abbildung 4.13 Modellgetriebene Generierung: Verhaltensbeschreibung Virtueller Prototyp

Um diesen letzten manuellen Schritt ebenfalls zu automatisieren, wurde eine skriptbasierte Generierung und damit eine Übertragung der Architekturinformationen, wie in Abbildung 4.14 gezeigt, gewählt. Eine detaillierte Beschreibung der skriptbasierten Lösung folgt in Abschnitt 5.3.2.

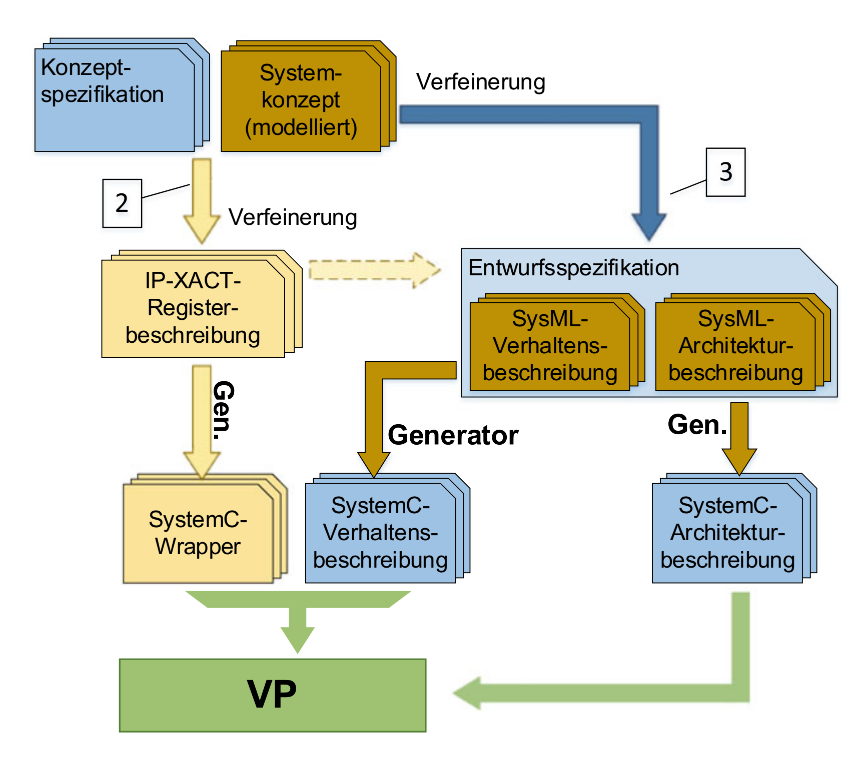


Abbildung 4.14 Modellgetriebene Generierung: Architektur Virtueller Prototyp

Anders als die Verhaltensbeschreibung ist die Architekturbeschreibung bereits im modellierten Systemkonzept auf System-Ebenen enthalten. Es muss lediglich eine Verfeinerung bzw. Präzisierung der modellierten Informationen, wie in Abschnitt 4.2.1 beschrieben, für die Generierung erfolgen. Die Generierung erfolgt in zwei Schritten. Im ersten Schritt werden alle benötigten Informationen der Architekturbeschreibung aus dem *Technical Model* vollautomatisiert in Tabellen übertragen. Anschließend werden die Informationen aus den erhaltenen Tabellen extrahiert und die Architekturinformationen in den Virtuellen Prototyp übertragen. Durch die hier gezeigte Erweiterung des teilautomatisierten Entwurf-Flows aus [15] um den modellgetriebenen Ansatz für die kontrollflussorientierten Module wird ein vollautomatisierter Generierungs-Flow für die Implementierung von Virtuellen Hardware-Prototypen ermöglicht.

4.3.2 Automatisierung Softwareentwurf

Wie in Abschnitt 2.5.1 beschrieben, wurde die UML als Softwarebeschreibungssprache entwickelt. Diese Fähigkeit blieb auch in der SysML im Kern erhalten und so lassen sich Software-Operationen und -Funktionen mittels Activity Diagrams bzw. Flowchart Diagrams darstellen und beschreiben. Aufgrund der steigenden Anforderungen und der daraus resultierenden Komplexität der SoCs gewinnt die On-Chip-Software, wie bereits in Abschnitt 2.6.5 beschrieben, immer stärker an Bedeutung. So werden immer mehr Funktionalitäten als Softwarecode auf Mikroprozessoren realisiert.

Wurde im vorangegangenen Abschnitt die Entwicklung der Virtuellen Prototypen automatisiert, um Softwarekonzepte bereits in einer früheren Phase der Entwicklungsphase testen zu können, soll nun die Softwareimplementierung selbst automatisiert werden. Durch die Automatisierung der Softwareentwicklung ist es möglich, den manuellen Aufwand in einer frühen Phase der Entwicklung zusätzlich zu senken, wie in Abbildung 4.15 veranschaulicht.

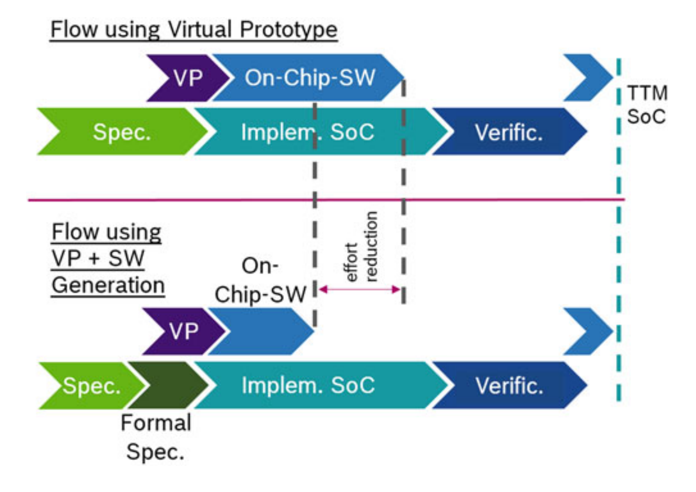


Abbildung 4.15 Automatisierter VP- und SW-Implementierungs-Flow

Neben der in der Abbildung 4.15 veranschaulichten erheblichen Reduzierung des Aufwandes durch die Generierung der On-Chip-Software beinhaltet dieser Schritt weitere entscheidende Vorteile. So wird durch die modellgetriebene Generierung der Software die Wahrscheinlichkeit von Fehlern reduziert. Dies gilt

besonders in Hinblick auf Inkonsistenzen zwischen modellierter Spezifikation und implementierter Software. Wird der hier entwickelte Softwaregenerierungs-Flow als Teil der Gesamtmethode in vollem Umfang eingesetzt, werden Änderungen am generierten Code lediglich durch Änderungen an dem Modell vorgenommen und der Code anschließend neu generiert. Dies ermöglicht die Sicherstellung einer zu jeder Zeit vollständigen und bezogen auf den generierten Code konsistenten Entwurfsspezifikation.

Diese Konsistenz der Spezifikation im aktuellen Entwicklungs-Flow zu gewährleisten, ist aufgrund der großen Entwicklungsteams, der langen Entwicklungszeiten und dem hohen Zeitdruck sehr schwierig. In der Entwurfsphase werden viele Entwurfsentscheidungen und Anpassungen im Zuge der Implementierung vorgenommen und direkt in Software bzw. Hardware umgesetzt. Die Dokumentation der Entwurfsentscheidungen und die Anpassungen in der Entwurfsspezifikation kommen dabei häufig zu kurz und es entstehen Inkonsistenzen.

Ein weiteres Problem bei der Spezifikation mittels natürlicher Sprache von Softwareverhalten besteht in der Schwierigkeit, diese adäquat und eindeutig in Fließtext als Systemanforderungen zu beschreiben. Darüber hinaus ist es sehr zeitaufwendig, bereits programmierten Softwarecode allein auf Basis des Codes nachzuvollziehen. Die Modellierung des Softwareverhaltens und die damit ermöglichte modellbasierte Entwicklung des Softwareentwurfs führt zu einer grafischen und in der Regel leichter verständlichen Beschreibung des Softwarecodes. Darüber hinaus ermöglicht die modellbasierte Entwicklung der Software eine Abstrahierung der Software-Funktionalität in verschiedene Ebenen und ermöglicht damit eine Arbeitsweise nach dem Top-down-Ansatz. So lässt sich, vergleichbar mit der System-Ebene bei der Hardware-Architektur-Entwicklung, die Software-Architektur im Modell als Teil des *Technical Model* abbilden und so das Zusammenspiel und die Kommunikation einzelner Funktionen darstellen. Dies ermöglicht nicht nur eine methodisch optimierte Softwareentwicklung, sondern erhöht das Gesamtverständnis aller beteiligten Entwickler/-innen und erleichtert damit die Kommunikation zwischen den Entwicklern/-innen und mit dem Kunden.

Die modellbasierte Generierung der Software erfolgt in C-Code mittels eines in dieser Arbeit konfigurierten Generators. Das Ergebnis der Generierung wird dabei maßgeblich durch zwei Faktoren beeinflusst. Zum einen hängt der resultierende Code von der Konfiguration des Generators und dem damit verbundenen Mapping zwischen SysML-Modellelement und C-Code ab. Zum anderen wird sie entscheidend durch die Modellierungstechnik beeinflusst. Auf die Konfiguration des Generators wird in Kapitel 5 näher eingegangen. Der Einfluss der Modellierung soll am Beispiel der Modellierung von Schleifen- und If-Abfragen

veranschaulicht werden. Dabei kommt das bereits vorgestellte Modellelement *Decision Node* zum Einsatz. Lediglich die Art der Verbindungen mittels *Activity Flow* entscheidet darüber, ob der Generator den modellierten Ablauf als Schleife oder If-Abfrage interpretiert und dementsprechend C-Code generiert. In Abbildung 4.16 werden die beiden Modellierungen gegenübergestellt.

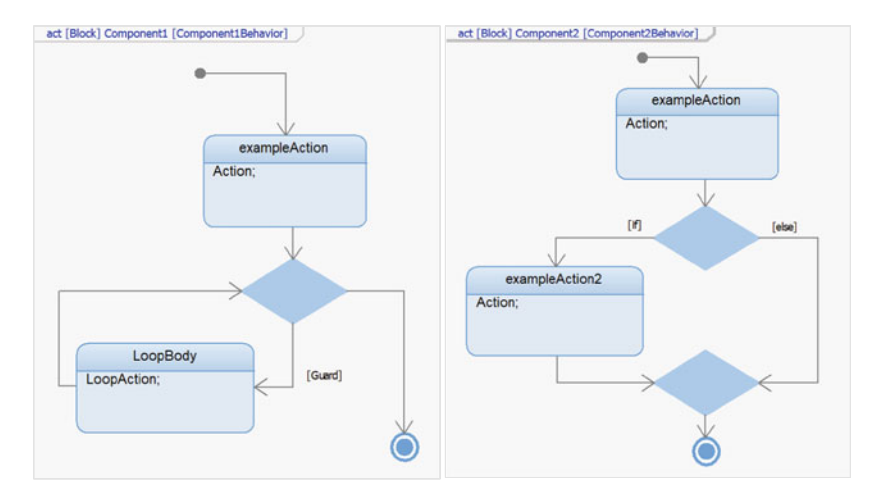


Abbildung 4.16 Vergleich Schleife- und If-Abfrage Modell

Links in Abbildung 4.16 ist eine While-Schleife als Beispiel dargestellt. Hierbei dient die *Decision Node* als Entscheidungsknoten. Der Actionblock *LoopBody* dient als Schleifenkörper und enthält die Anweisungen, wie zum Beispiel das Hochzählen einer Zählervariablen, welche in der Schleife ausgeführt werden sollen. Der *Guard* am *Activity Flow,* welcher den Entscheidungsknoten mit dem *LoopBody* verbindet, dient als Schleifenbedingung für den Entscheidungsknoten. Ist die Schleifenbedingung nicht erfüllt, gilt der zweite Weg vom Entscheidungsknoten zum Terminationspunkt; dies ist gleichzeitig das Funktionsende.

Rechts in der Abbildung 4.16 ist die Modellierung einer If-Abfrage bzw. in diesem Fall einer If-Else-Abfrage abgebildet. Zentrales Modellelement ist wieder die *Decision Node* als Entscheidungsknoten. Ist die If-Bedingung des linken Entscheidungspfades erfüllt, wird die Anweisung des Actionblocks *exampleAction2* ausgeführt. Ist die If-Bedingung nicht erfüllt, greift der Else-Entscheidungspfad und der Actionblock wird übergangen. Die beiden Entscheidungspfade laufen

wieder zusammen in einem Modellelement, welches in der Darstellung der *Decision Node* deckungsgleich ist. Jedoch handelt es sich hierbei um einen *Merge Node,* der die Aufgabe erfüllt, Entscheidungsknoten zusammenzuführen. Beide Entscheidungspfade enden im selben Terminationspunkt, was auch hier dem Funktionsende entspricht.

4.3.3 Modellgetriebene Verifikation

Durch die Zunahme in Umfang und Komplexität der Implementierung und die damit einhergehende Heterogenität von Automotive SoCs wächst die Verifikation dieser Systeme zu einer herausfordernden Aufgabe für die SoC-Entwicklung. Die Aufwände für die Verifikation komplexer Systeme übersteigen dabei, wie in der Fallstudie in [84] gezeigt, stellenweise die Aufwände für den Entwurf selbst.

So ist eine hohe Qualität in Form von Fehlerfreiheit, Robustheit und Zuverlässigkeit entscheidend bei der Entwicklung von Automotive SoCs, welche beispielsweise im Bereich der Fahrassistenzsysteme und der Fahrsicherheitssysteme zum Einsatz kommen. Um dies sicherzustellen, bedarf es einer umfangreichen Verifikation [2]. Diese zu planen, zu entwickeln und durchzuführen führt zu erheblichen Kosten- und Zeitaufwänden und schränkt nach heutigen Methoden die Effizienz der SoC-Entwicklung ein. Daher sollen als Teil der in dieser Arbeit entwickelten modellbasierten Entwicklungsmethode Möglichkeiten der Aufwandsreduzierung in der Verifikation betrachtet werden. Dabei werden zwei Aspekte der Verifikation maßgeblich durch die hier entwickelte Gesamtmethode aufgegriffen:

- **Reduzierung der Spezifikationsfehler** durch Modellierung und Formalisierung der Spezifikation
- **Automatisierung der Verifikationserstellung** auf Grundlage der modellierten Spezifikation

Reduzierung der Spezifikationsfehler

Wie in Abschnitt 2.6.8 beschrieben, wird von Spezifikationsfehlern im Falle einer falschen, nicht eindeutigen oder nicht vollständigen Spezifikation gesprochen. Nicht vollständig meint hierbei, dass ein mögliches Szenario, welches im späteren Betrieb auftreten kann, in der Spezifikation nicht erfasst wurde. Dabei führt besonders die fehlende Eindeutigkeit der Spezifikation zu weitreichenden Problemen. So sind aus diesem Grund die Entwickler/-innen beim Implementieren des Entwurfes wie

auch die Verifikationsingenieure/-innen bei der Verifikationserstellung gezwungen, die meist in natürlicher Sprache verfassten Anforderungen zu interpretieren. Durch die Interpretation der Anforderungen in der Spezifikation entsteht ein eigenes Bild des Systems in der Vorstellung der Entwickler/-innen, welches nur die Entwickler/-innen selbst kennen. Neben der Gefahr von Fehlern erschwert somit die Interpretierbarkeit der natürlichen Sprache die Kommunikation sowohl zwischen den Entwicklern/-innen als auch zwischen Kunde und Entwickler.

Zusätzlich zur fehlenden Eindeutigkeit der Spezifikation führt besonders die fehlende Vollständigkeit zu einer Minderung der Effizienz in der Verifikation. Wird das Fehlen von Informationen beispielsweise während der Erstellung der Verifikation entdeckt, müssen die fehlenden Informationen mit hohem Aufwand nachträglich eingeholt werden. Hierbei lässt sich, ähnlich wie bei der Interpretierbarkeit, der Vergleich zu den Entwicklern/-innen ziehen, welche ein System in ihrem Kopf weiterentwickeln, ohne dieses jedoch für alle ersichtlich zu spezifizieren. So neigen in der SoC-Entwicklung besonders die SoC-Architekt/innen als fachliche Leiter des Entwicklungsprojektes aufgrund ihrer Erfahrungen und Vorkenntnisse dazu, trivial erscheinende Informationen nicht zu spezifizieren. Hinzu kommt die Gefahr, dass die Verifikationsingenieure/-innen aufgrund der fehlenden Informationen in der Spezifikation sich diese durch einen Blick in den implementierten Entwurf einholen und somit die Verifikation, wie in Abschnitt 2.6.8 beschrieben, an Wirkung verliert. Wird das Fehlen von Informationen in der Spezifikation bei der Erstellung der Verifikation jedoch nicht entdeckt, werden Teile des Systems nicht vollständig verifiziert und mögliche Fehler nicht entdeckt. Dennoch werden 79 % der Anforderungen heutiger Spezifikationen, wie in [85] gezeigt, mittels interpretierbarer, schwer validierbarer und fehleranfälliger natürlicher Sprache spezifiziert.

Die hier entwickelte Gesamtmethode und die darin enthaltene Modellierungsmethode ermöglicht durch die grafische und formalisierte Modellierung des SoCs eine Steigerung der Eindeutigkeit der Spezifikation. Somit werden Interpretationen bei der Erstellung des Entwurfs und damit Inkonsistenzen zwischen Entwurf und Spezifikation minimiert. Bezogen auf die Verifikation sinkt hierdurch zwar nicht der Aufwand, jedoch steigt die Qualität der Verifikation und damit das Ergebnis bei gleichbleibendem Aufwand, wodurch die Effizienz der Verifikation gesamt gesteigert werden kann. Für die Erstellung der Verifikation bedeutet die durch die hier entwickelte Methode hinzugewonnene Eindeutigkeit zudem eine Minimierung des Bedarfs der Interpretation, welcher bei der Übertragung der Informationen aus der Spezifikation entsteht. So können Fehler bzw. Inkonsistenzen, welche bei der Erstellung der Verifikation selbst entstehen, reduziert werden.

Lässt sich dies auch schwer belegen oder beweisen, so kann dennoch davon ausgegangen werden, dass durch die Nähe der grafischen Modellierung zur bildbasierten Denkweise der Entwickler/-innen die Validierung der grafisch modellierten Spezifikation auf Vollständigkeit und Fehlerfreiheit deutlich leichter fällt als die Validierung von natürlicher Sprache. Dies führt somit zu einer Steigerung der Vollständigkeit der Spezifikation und damit zu einer Reduzierung des Aufwandes, welcher benötigt wird, um fehlende bzw. falsche Informationen nachträglich einzuholen. Wie gezeigt, minimiert bereits die Formalisierung und Modellierung der Spezifikation maßgeblich aktuelle Defizite der Verifikation von Automotive SoCs und ermöglicht dadurch eine Steigerung der Effizienz im Bereich der Verifikation.

Automatisierung der Verifikationserstellung

Da wie bereits erwähnt der Aufwand für die Verifikation komplexer Systeme in manchen Fällen die Aufwände des Entwurfs übersteigt, aber in jeder Hinsicht einen erheblichen Anteil der Entwicklung ausmacht, soll als Teil der hier entwickelten Methode die Automatisierung der Verifikationserstellung ermöglicht und nachfolgend konzeptuell behandelt werden [86].

Neben einer erheblichen Reduzierung des Aufwandes birgt die Automatisierung der Verifikationserstellung und die damit einhergehende Steigerung der Effizienz weitere entscheidende Vorteile. So wird durch die Automatisierung der Übertragung bzw. die Transformation der Informationen aus der Spezifikation in die Verifikation die Fehlerwahrscheinlichkeit erheblich minimiert. Ein manueller Übertrag beinhaltet neben der Gefahr von falschen Interpretationen stets die Gefahr von einfachen Tippfehlern und Implementierungsfehlern bei der Erstellung der Verifikation.

Wie bei der manuellen Erstellung der Verifikation muss auch für die Automatisierung der Verifikationserstellung das „Vier-Augen-Prinzip“ gelten. Bei der manuellen Entwicklung bedeutet dies, wie in Abschnitt 2.6.8 beschrieben, dass die Entwickler/-innen des Entwurfs und die Entwickler/-innen der Verifikation zwei unabhängig voneinander arbeitende Personen sein müssen. Sollen, wie in dieser Arbeit beschrieben, Teile des Entwurfs wie auch Teile der Verifikation auf Basis der modellierten Spezifikation generiert werden, besteht die Möglichkeit, dass der generierte Teil des Entwurfs vom generierten Teil der Verifikation verifiziert wird. Damit steigt das Risiko von nicht entdeckten sogenannten „Common Cause Failures“ (häufige Fehlerursachen).

Das „Vier-Augen-Prinzip“ ist jedoch für die Automatisierung erfüllt, wenn zwei vollkommen unabhängig entwickelte und arbeitende Generatoren bzw. Generierungsmethoden für Entwurf und Verifikation verwendet werden. Durch die Verwendung zweier unabhängiger Generatoren erhält man die höchste Wahrscheinlichkeit, dass nicht derselbe Fehler, welcher bei der Generierung des Entwurfs

entstanden ist, auch bei der Generierung für die Verifikation entsteht. Somit wird als Gegenstück des „Vier-Augen-Prinzips“ der Implementierung das „Vier-Augen-Prinzip“ der Generierung erzeugt. Außerdem muss beachtet werden, dass ein Generator in der Regel auf ein vordefiniertes Mapping zwischen Modell und Programmcode zurückgreift. Die Definition dieses Mappings sollte für den jeweiligen Generator in jedem Fall unabhängig erfolgen [13].

Für die vorliegende Arbeit wird in Kapitel 5 exemplarisch ein „Proof-of-Concept“ für die Automatisierung der Verifikationserstellung anhand von „Connectivity Checks“ (Konnektivitätsprüfungen) behandelt. Dazu erfolgt die Generierung der „Connectivity Checks“ als Teil der Verifikation aus dem *Architecture Diagram.* „Connectivity Checks“ überprüfen auf Grundlage der Spezifikation die korrekte „Verdrahtung“ des SoCs im Entwurf. Für die Generierung der „Connectivity Checks“ werden die Informationen über die Verbindungen zwischen den Modulen aus dem *Architecture Diagram* skriptbasiert ausgelesen und in SystemVerilog-Assertions übersetzt. Mithilfe der weitverbreiteten Beschreibungssprache SystemVerilog lassen sich Teile der Verifikation beschreiben. Dabei werden Assertions auf Grundlage der Spezifikation definiert, um zu prüfen, ob der implementierte Entwurf die Aussagen aus der Spezifikation erfüllt [87]. Die Automatisierung der Entwicklung von „Connectivity Checks“ auf Basis des SysML-Modells zeigt die Möglichkeiten der Effizienzsteigerung im Bereich der Verifikationserstellung auf.

4.3.4 Zwischenergebnis

In diesem Abschnitt wurden Möglichkeiten der modellgetriebenen Automatisierung für den Systementwurf betrachtet. Dazu wurde die Generierung der Architekturbeschreibung wie auch der Verhaltensbeschreibung des Hardware-VPs konzeptuell behandelt, und die Vorteile der Automatisierung wurden diskutiert. Darüber hinaus wurde die Generierung der On-Chip-Software als Teil der hier beschriebenen Methode vorgestellt sowie die Automatisierung der Verifikationserstellung behandelt. Die Automatisierung als Teil der Implementierung ist in Abbildung 4.17 als Übergang zwischen Systemmodell und den Bereichen Software, Hardware-VP (Virtueller Prototyp) und Verifikation dargestellt.

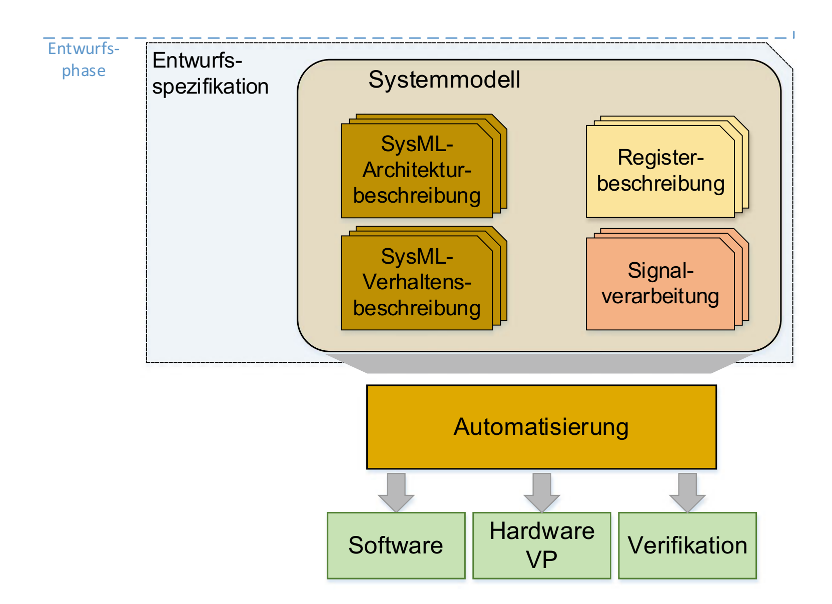


Abbildung 4.17 Zwischenergebnis modellgetriebener Systementwurf

Die Generierung erfolgt bei den hier behandelten Ansätzen auf Basis der SysML-Architektur- bzw. der Verhaltensbeschreibung des Systemmodells. Durch die modellgetriebene Generierung von Code entsteht die Möglichkeit einer Arbeitsweise, welche Änderungen am implementierten Entwurf nur durch Änderungen am Modell erlaubt. Dies ermöglicht somit eine nahezu vollständige Konsistenz zwischen modellierter Spezifikation und implementiertem Code und unterstützt damit die Erfüllung der Anforderung A3. Durch die Automatisierung selbst kann gemäß Anforderung **A7** der Aufwand bei der Implementierung für die jeweiligen Bereiche der Entwicklung reduziert werden. Darüber hinaus wird durch die Modellierung und Formalisierung der Spezifikation verbunden mit der Automatisierung eine Reduzierung des Aufwandes in der Verifikationserstellung (Anforderung **A8**) ermöglicht.

4.4 Zwischenergebnis

In diesem Kapitel wurde die modellbasierte Entwicklungsmethode vorgestellt. Abbildung 4.18 veranschaulicht den angepassten Entwicklungs-Flow in Gesamtheit unter Anwendung der in dieser Arbeit entwickelten Methode. Das Systemkonzept und das daraus resultierende Systemmodell wurden in der Abbildung 4.18 abstrahiert. Sie bestehen jeweils aus der Kombination von SysML, IP-XACT und signalverarbeitender Beschreibung.

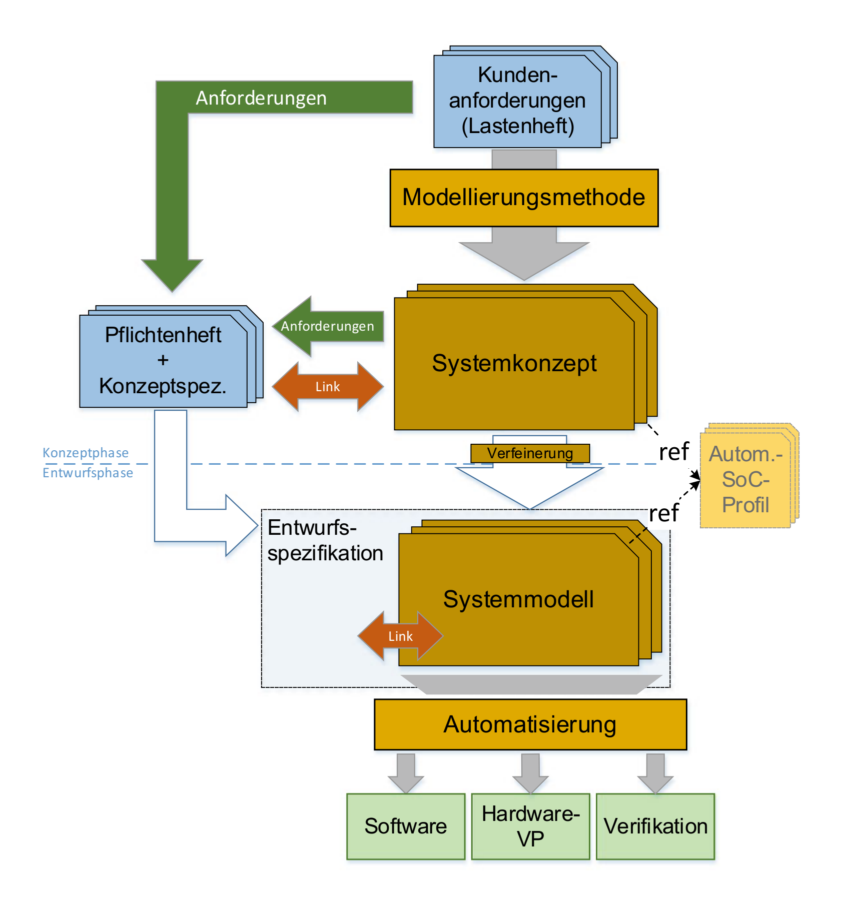


Abbildung 4.18 Zwischenergebnis Konzept

Durch die Definition der in Abschnitt 4.1 beschriebenen Modellierungsmethode für die Realisierung einer modellbasierten Systemkonzeptentwicklung und die damit verbundene Formalisierung und Modellierung der Spezifikation kann die Wahrscheinlichkeit von nachträglichen Änderungen an den zwischen Kunde und Entwickler vereinbarten Anforderungen reduziert werden. Zudem wurde die Wahrscheinlichkeit von Implementierungsfehlern aufgrund einer unzureichenden Spezifikation gesenkt. Die Fehlerbehebung sowie die Änderungen an den vereinbarten Anforderungen führt oftmals zu hohen Aufwänden und damit zu steigenden Kosten bei der Entwicklung. Werden zudem die Fehler erst in einer sehr späten Phase des Projekts entdeckt oder müssen aufgrund von nicht ausreichend analysierten Kundenanforderungen diese nachträglich angepasst werden, kann es zu Verzögerungen im Projekt oder einer Verschlechterung der Qualität kommen. Die vorgestellte Modellierungsmethode erhöht somit die Effizienz sowohl bei der Konzeptentwicklung als auch bei der Implementierung.

Die Methode wendet den modellbasierten Ansatz über den gesamten Entwicklungs-Flow an, das heißt, von den Kundenanforderungen über das Systemkonzept bis hin zur Entwurfsspezifikation. Durch diesen ganzheitlichen Ansatz wird ein hohes Maß an Konsistenz in der Spezifikation in allen Phasen der Entwicklung ermöglicht und die Nachverfolgbarkeit bei den Anforderungen erreicht. Zudem findet eine Integration der hier entwickelten Methode in den aktuell bestehenden SoC-Entwicklungs-Flow und die Verlinkung unterschiedlicher Beschreibungsformen statt. Durch die Verlinkung der Beschreibungsformen werden zusätzlich Inkonsistenzen zwischen unterschiedlichen Spezifikationsformaten reduziert. Inkonsistenzen in der Spezifikation führen zu Inkonsistenzen zwischen der Spezifikation und dem implementierten Entwurf. Somit kann die Anwendung der Methode die Aufwände für die dadurch benötigten Änderungen sowohl an der Spezifikation wie auch an der Implementierung senken. Die Minimierung von Inkonsistenzen führt somit zu einer höheren Qualität des Systems, reduziert die Kosten und verhindert Verzögerungen, welche durch Änderungen in späten Phasen der Entwicklung entstehen. Es findet daher erneut eine Effizienzsteigerung bei der Entwicklung durch die Integration der Methode in den aktuellen SoC-Entwicklungs-Flow statt.

Zur Bereitstellung von Templates und der damit verbundenen Aufwandsreduzierung bei der Modellierung sowie der Steigerung der Anwenderfreundlichkeit wurde das Automotive-SoC-Profil entwickelt. Dabei wird durch das Profil eine Vereinheitlichung der modellierten Spezifikation über die Projekte hinweg erreicht und die Entwickler/-innen werden an eine Top-down-basierte Arbeitsweise gebunden.

Der Übergang von Konzeptphase zu Entwurfsphase und die damit einhergehende Verfeinerung des Systemmodells über die System-Ebene hinaus wurde in Abschnitt 4.2 behandelt. Die Verfeinerung legt dabei den Grundstein für die in Abschnitt 4.3 beschriebene Automatisierung der Entwicklung von Virtuellen Hardware-Prototypen sowie der Software. Die Automatisierung von Entwicklungsprozessen führt nicht nur zu einer Reduktion des manuellen Aufwandes und damit der Entwicklungskosten. Es wird zudem eine gleichbleibend hohe Qualität des Entwurfs ermöglicht und somit Verzögerungen in der Entwicklung vorgebeugt. Die aufgezeigten Automatisierungen führen somit zu einer erhöhten Effizienz, auch in der Entwurfsphase.

Darüber hinaus wurde die modellgetriebene Automatisierung, wie in Abschnitt 4.3.3 beschrieben, auch auf den Bereich der Verifikationserstellung angewendet. Dabei wurden sowohl die Vorteile der modellbasierten Spezifikation für die Qualität der Verifikation diskutiert als auch die Automatisierung der Verifikationserstellung selbst an einem Beispiel behandelt.

Die in diesem Kapitel behandelten Teile dieser Arbeit setzen sich zu einer modellbasierten Entwicklungsmethode für die SoC-Entwicklung zusammen, welche dadurch die Steigerung der Effizienz über den gesamten SoC-Entwicklungs-Flow ermöglichen kann. Die Implementierung der modellbasierten Entwicklungsmethode sowie die Anwendung dieser auf industrielle SoC-Beispiele wird im nachfolgenden Kapitel 5 beschrieben. Die Evaluierung der hier beschriebenen Entwicklungsmethode und damit die Überprüfung, ob diese die in dieser Arbeit gestellten Anforderungen erfüllt, folgt in Kapitel 6.



5 Implementierung

Als Teil des folgenden Kapitels wird die Implementierung der in Kapitel 4 konzeptuell beschriebenen modellbasierten Entwicklungsmethode anhand von zwei SoC-Beispielen behandelt und veranschaulicht.

- Dabei wird die Methode für die modellbasierte Systemkonzeptentwicklung auf ein Beispiel eines Auswerte-SoCs in der Konzeptphase angewendet und so die Modellierung und Formalisierung der Spezifikation veranschaulicht.
- Für die Behandlung der Implementierung der modellbasierten Entwicklungsmethode in der Entwurfsphase und der damit verbundenen modellgetriebenen Automatisierung des Entwurfs dient ein zweites SoC-Beispiel.

Zu Beginn von Abschnitt 5.1 wird die für die Implementierung der modellbasierten Entwicklungsmethode genutzte Tool-Umgebung vorgestellt und auf deren Nutzen für die hier vorliegende Arbeit eingegangen. Darüber hinaus werden in Abschnitt 5.1 die Implementierung der Modellierungsmethode und die Formalisierung im Kontext der modellbasierten Systementwicklung behandelt. In Abschnitt 5.2 folgt die Beschreibung des Prozesses der Modellverfeinerung als Teil des Übergangs zwischen Konzept- und Entwurfsphase. Die modellgetriebene Automatisierung des Entwurfs und der Verifikationserstellung werden in Abschnitt 5.3 vorgestellt.

5.1 Modellbasierte Systementwicklung

In den nachfolgenden Abschnitten wird die Anwendung der SysML-basierten Modellierungsmethode anhand eines Beispiel-Auswerte-SOCs behandelt. Dabei

A. Kirchner, *Entwicklung von Methoden zur abstrakten Modellierung von Automotive Systems-on-Chips*, https://doi.org/10.1007/978-3-658-38437-1_5

wird die Modellierung eines SoCs in der Konzeptphase durch Diagrammbeispiele veranschaulicht. Darüber hinaus wird die Verlinkung zwischen den Tools IBM® Rational® Doors® Next Generation und IBM® Rational® Rhapsody® als Teil des Beispiels behandelt, ebenso die Möglichkeit der Automatisierung bei der Modellierung.

5.1.1 Definition Tool-Umgebung

Die in Kapitel 4 vorgestellte modellbasierte Entwicklungsmethode baut in ihrer Implementierung auf verschiedenen Tools aus den Bereichen der Eingebetteten Systeme und der Systementwicklung auf. Daher werden nachfolgend die für die Umsetzung der Methode essenziellen Tools vorgestellt und der Nutzen für die hier vorliegende Methode beschrieben.

IBM® Rational® Rhapsody®
IBM® Rational® Rhapsody® (Rhapsody) ist eine Entwicklungsumgebung für die modellbasierte Entwicklung von Eingebetteten Systemen sowie von Software. Dabei unterstützt Rhapsody neben vielen anderen standardisierten Modellierungssprachen die in dieser Arbeit genutzte SysML und ermöglicht darüber hinaus die Simulation dieser Modelle. Zusätzlich zur Simulation bietet Rhapsody verschiedene Standard-Codegeneratoren, welche auf Basis der SysML-Diagramme die Generierung von Code, wie zum Beispiel C und C++, unterstützen. Diese Generatoren lassen sich durch die Anwender/-innen entsprechend ihren Anforderungen und Anwendungsfällen konfigurieren. Weiter bietet Rhapsody eine Vielzahl an vordefinierten Profilen an. So gibt es für jede Modellierungssprache eigene Profile, um die Modellierungsoberfläche an die entsprechende Sprache bzw. die Anwender/-innen anzupassen [59].

Nachfolgend sollen wichtige genutzte Funktionalitäten des Tools behandelt werden, welche einen direkten Einfluss auf die Implementierung der in dieser Arbeit entwickelten Methode haben.

Properties
Rhapsody bietet die Möglichkeit, eine Vielzahl der Eigenschaften der Nutzeroberfläche, der Modellierung und der Codegenerierung über sogenannte „Properties“ (Eigenschaften) anzupassen. Als Properties werden durch den Nutzer änderbare Variablen bezeichnet, über die nahezu jedes Element in Rhapsody verfügt. Die Definitionen der Properties können auf unterschiedlichen Ebenen vorgenommen

werden. So lassen sich Properties für das gesamte Modell wie auch für einzelne Modellelemente eines Diagramms konfigurieren. Dabei stehen sie in einer Vererbungsbeziehung, d. h., Änderungen auf niedriger Ebene (Modellelement) überschreiben Properties auf Modellebene [59].

Ein Beispiel für den Einsatz der „Properties" ist die Definition von grafischen Eigenschaften von Modellelementen und Diagrammen als Teil eines Profils. Wird das Profil in das Modell geladen, werden gleichzeitig die hier definierten Werte für die „Properties" übernommen und angewendet. So wurden auch als Teil des in dieser Arbeit entwickelten Automotive-SoC-Profils „Properties" definiert, welche für alle Modelle der SoC-Entwicklung gleichermaßen benötigt werden. Durch die Definition von „Properties" für die Darstellung der Modellelemente und Diagramme als Teil des Automotive-SoC-Profils kann eine einheitliche Darstellung in allen Modellen verschiedener SoC-Projekte geschaffen werden.

Tags

Tags stellen eine weitverbreitete Ergänzung zu den in SysML enthaltenen Variablen dar. Während Variablen in erster Linie zur Spezifikation von funktionalen Eigenschaften im Modell vorgesehen sind, lassen sich mittels Tags beliebige Eigenschaften beschreiben. Dabei lassen sich Tags wie Variablen verwenden und beliebigen Modellelementen zuweisen. Sie unterliegen dabei jedoch nicht den Beschränkungen der Variablen [59]. In der hier vorliegenden Arbeit werden Tags in erster Linie zur Spezifikation von nichtfunktionalen Eigenschaften des Systems im Modell verwendet, welche nicht für eine spätere Simulation benötigt werden. Dazu wurden auf Grundlage eines „Stakeholder"-Interviews ermittelte Eigenschaften mittels Tags vordefiniert, in Stereotypen gruppiert und den Anwender/-innen als Teil des Automotive-SoC-Profils zur Verfügung gestellt.

Rhapsody Java API

Neben den Standardfunktionalitäten von Rhapsody bietet das Tool die Möglichkeit, über eine Java API mittels Plug-In (Software-Erweiterung) zusätzliche Funktionen zu realisieren. So besteht die Möglichkeit, Applikationen in Rhapsody zu integrieren und mittels dieser beispielsweise „Properties" zu manipulieren, die Modellierung selbst zu automatisieren oder Informationen aus bestehenden Diagrammen auszulesen [59]. Die Java API wurde für die hier vorliegende Arbeit sowohl zur Automatisierung der Modellierung verwendet, um so die Anwenderfreundlichkeit zu erhöhen und den Modellierungsaufwand zu senken, als auch um darüber hinaus mittels Plug-Ins die Informationen der Architektur für die Generierung der „Connectivity Checks" auszulesen. Eine nähere Beschreibung dieses Plug-Ins folgt in Abschnitt 5.3.4.

IBM® Rational® Doors® Next Generation

IBM® Rational® Doors® Next Generation (DNG) ist ein webbrowserbasiertes Anforderungsmanagement-Tool zur Dokumentation und Verwaltung von Anforderungen während des gesamten Entwicklungs-Flows. Dabei ermöglicht DNG die Strukturierung der Anforderungen in Form von Artefakten, welche jeweils über eine eindeutige Identifikationsnummer verfügen. Neben der Beschreibung der reinen Anforderungen lassen sich durch die Anwender/-innen Attribute und so beispielsweise Sollwerte oder Toleranzgrenzen den Artefakten zuweisen. Die eindeutige Identifikationsnummer der Anforderungen ermöglicht zudem das Setzen von Verlinkungen zwischen den Anforderungen und realisiert damit eine Nachverfolgbarkeit über den Entwicklungs-Flow hinweg [88].

DNG ist wie Rhapsody Teil der IBM-Rational-Tool-Umgebung und wird in der hier vorliegenden Arbeit für die Verwaltung der Systemanforderungen genutzt, da es neben der Verlinkung der Anforderungen selbst die Verlinkung der Anforderungen mit Artefakten anderer Tools der IBM-Rational-Tool-Umgebung ermöglicht. Somit wird eine direkte Verlinkung zwischen den Anforderungen in DNG und den Modellelementen in Rhapsody ermöglicht [88].

IBM® Rational® Rhapsody® Model Manager

Der IBM® Rational® Rhapsody® Model Manager (RMM) dient zur serverbasierten Speicherung und Verwaltung von Modellen, welche in Rhapsody erstellt wurden. Darüber hinaus ermöglicht die Nutzung des RMM die Verlinkung von Modellelementen in Rhapsody und den Systemanforderungen in DNG. Bei den Verlinkungen handelt es sich um Open Services for Lifecycle Collaboration (OSLC)-Verknüpfungen. OSLC ist eine offene Initiative, welche sich mit der Verlinkung und Integration von Spezifikation und unterschiedlichen Werkzeugen beschäftigt, um durch ein Standardformat den Austausch zu erleichtern [89].

Neben der Verlinkung mit anderen Tools der Rational-Tool-Umgebung bietet die Speicherung der Modelle auf dem RMM-Server den Vorteil, dass beliebig viele Entwickler/-innen parallel am selben Modell arbeiten können. Dazu erzeugen sich die Entwickler/-innen ein lokales Abbild des Modells und bearbeiten dieses. Anschließend werden die Änderungen auf den Server hochgeladen. Dabei werden die Änderungen an den entsprechenden Diagrammen und Modellelementen selektiv auf den Server übertragen, ohne dabei die unveränderten Anteile zu überschreiben. Lediglich bei einer gleichzeitigen Änderung desselben Modellelements müssen die Anwender/-innen selbst aktiv werden [59].

5.1.2 Modellierungsmethode

Im folgenden Abschnitt wird die Anwendung der in dieser Arbeit entwickelten modellbasierten Entwicklungsmethode auf die Systemkonzeptentwicklung anhand eines Beispiel-Auswerte-SoCs beschrieben. Dazu wurde im Zuge der hier vorliegenden Arbeit die Modellierung des Auswerte-SoCs durchgeführt. Die so erhaltenen Diagramme sollen nachfolgend zur Veranschaulichung der modellbasierten Entwicklungsmethode dienen.

Eine Sonderrolle erhält hierbei das Package Diagram, da es Teil der in dieser Arbeit entwickelten Modellierungsmethode, jedoch nicht Teil der Formalisierung ist. Grund hierfür ist die Verwendung des Package Diagram rein zur Modellierung der Modellstruktur selbst. Es wird somit nicht zur Modellierung des SoCs beziehungsweise dessen Umfelds verwendet. Die Modellierung der Modellstruktur als Package Diagram dient in erster Linie zur schnelleren Navigation im Modell, da die Anwender/-innen hierdurch zum einen Informationen über die Struktur des Modells enthalten und zum anderen über Verlinkungen schnell in die entsprechenden Diagramme des Modells navigieren können. Abbildung 5.1 zeigt die in der hier vorliegenden Arbeit entwickelte und in Abbildung 4.4 dargestellte Modellstruktur, angewendet auf den in diesem Abschnitt zugrunde liegenden Auswerte-SoC als Beispiel.

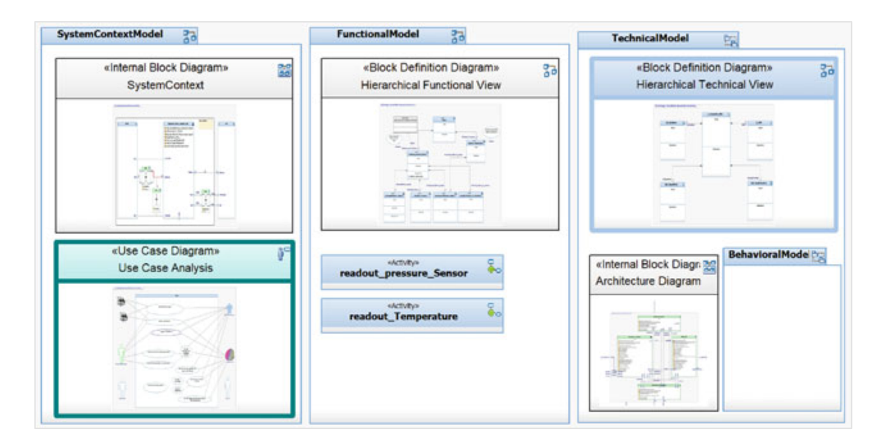


Abbildung 5.1 Modellstruktur Beispiel-Auswerte-SoC Konzeptphase

Das Package Diagram genannt „*StartPage*“ zeigt die in dieser Arbeit definierte Modellstruktur anhand der Packages *System Context Model*, *Functional Model*, *Technical Model* und *Behavioral Model* und stellt zudem die wichtigsten Diagramme des jeweiligen Submodells als Block, inklusive Abbildung des Diagramms, dar. Die Abbildungen der Diagramme dienen dabei lediglich als Orientierungshilfe und werden im Verlauf der Arbeit näher betrachtet. Über die in der oberen rechten Ecke der Blöcke enthaltenen Buttons wird es den Anwendern/-innen ermöglicht, direkt zu dem jeweiligen Diagramm zu navigieren. Die finale Modellierung einer solchen *StartPage* ist in der Regel erst zu einem späteren Zeitpunkt der Konzeptphase möglich. Eine generalisierte Version wird den Anwendern/-innen jedoch als Teil des Automotive-SoC-Profils zur Verfügung stehen. Die Zurverfügungstellung der Modellstruktur als Teil des in dieser Arbeit entwickelten Automotive-SoC-Profils bietet dabei folgende Vorteile:

- Formalisierung der Modelle über alle SoC-Entwicklungsprojekte
- Reduzierung des Aufwandes und der Fehleranfälligkeit bei der Modellierung
- Unterstützung des Anwenders in der Entwicklung nach dem Top-down-Ansatz

Die modellbasierte Entwicklung des SoCs beginnt, wie in Kapitel 4 beschrieben, in der Regel mit der Analyse der Anwendungsfälle und damit der Kundenforderungen. Hierzu dient das in Abbildung 5.2 dargestellte Use Case Diagram, genannt *Use Case Analysis* als Beispiel.

Auf der linken Seite des hier dargestellten Use Case Diagram sind die Systeme des übergeordneten Systems zu sehen, welche mit dem Auswerte-SoC interagieren bzw. mit denen der Auswerte-SoC interagiert. So wurde beispielsweise als Teil des hier gezeigten Use Case Diagram der Anwendungsfall „readout pressure signal“ modelliert. Hierbei wird das Drucksignal von einer der in der Abbildung 5.2 dargestellten „Engine Control Units“ (ECUs, Motorsteuergeräte) vom SoC angefordert. Um dieses Drucksignal an die ECU liefern zu können, benötigt der SoC wiederum eine Interaktion mit dem Sensormodul, welches den aktuellen Druck misst.

Neben den Systemen des übergeordneten Systems wurden in diesem Beispiel auf der rechten Seite des Diagramms bereits Schnittstellen zur Fehleranalyse und zum Wafer-Level-Test dargestellt und die dafür benötigten Anwendungsfälle analysiert. Die Abbildung 5.2 veranschaulicht zudem die Möglichkeit, die Standardsymbole des Use Case Diagram durch eigene Symbole zu ersetzen. Diese Funktion lässt sich auf beliebige Modellelemente in allen Diagrammen anwenden. Die Nutzung sollte sich jedoch auf das Use Case Diagram beschränken, um in Diagrammen wie BDDs und IBDs weiterhin eine grafische Wiedererkennbarkeit der Modellelemente zu erhalten.

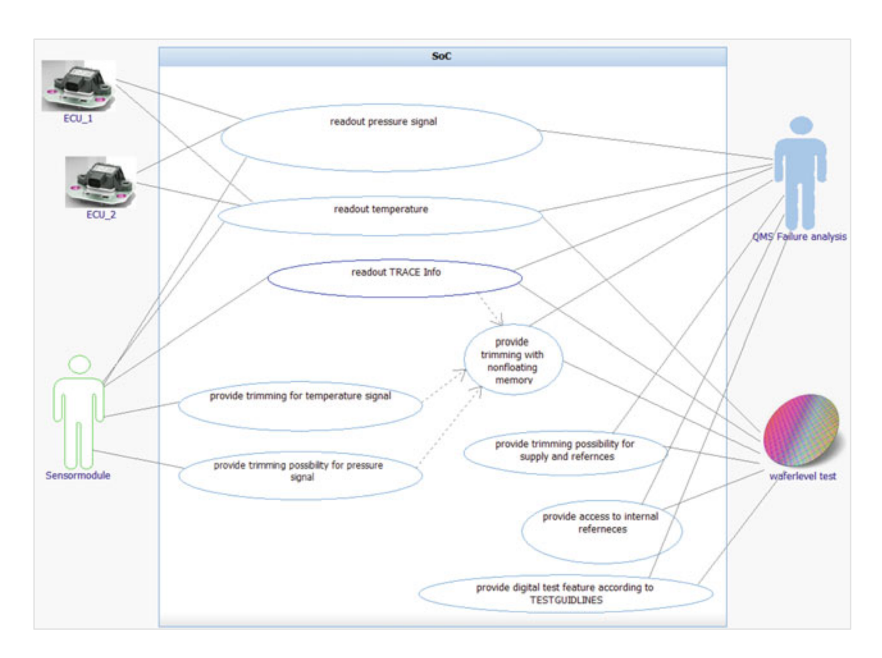


Abbildung 5.2 Use Case Diagram Beispiel-Auswerte-SoC

Zusätzlich zur Analyse des Anwendungsfalls wurden eine Analyse und eine Modellierung des technischen Systemkontexts durchgeführt, deren Ergebnis in Abbildung 5.3 dargestellt ist.

Das *System Context Diagram* zeigt den Auswerte-SoC als Part in der Mitte des Diagramms sowie die externen Systeme, welche mit dem Auswerte-SoC interagieren. Wenn möglich, sollte hier die physikalische Anordnung der Systeme im übergeordneten System eingehalten werden. Neben den Systemen beinhaltet das *System Context Diagram* die Darstellung der Ports und spezifiziert bereits einige Eigenschaften des Auswerte-SoCs mittels Tags. Darüber hinaus wurden in diesem Diagramm Standardelemente, in diesem Fall Kondensatoren modelliert, um die Beschaltung des Auswerte-SoCs zu veranschaulichen. Diese Elemente wurden, wie in Abschnitt 4.1.3, als vorgefertigte Templates aus dem in dieser Arbeit entwickelten Automotive-SoC-Profil in das hier gezeigte Diagramm kopiert. Zudem wurden den in Abbildung 5.3 enthaltenen Parts vordefinierte Ports aus dem SoC-Profil zugeordnet sowie dem Part *itsAuswerteSoC* die Tags *ASIL_level* und *Vendor* hinzugefügt.

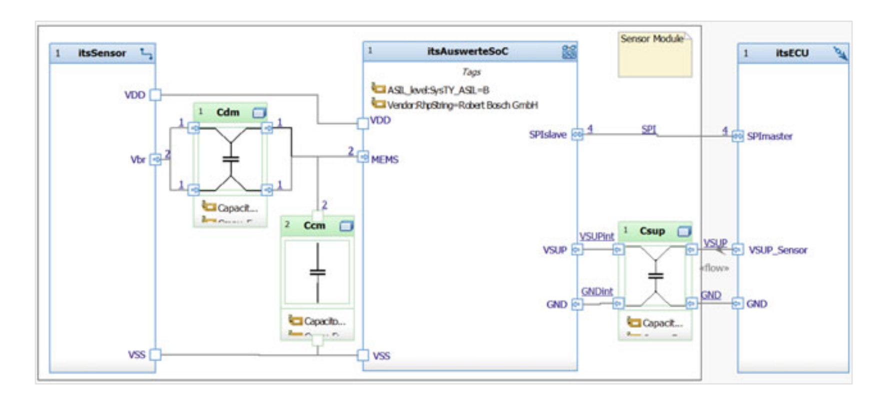


Abbildung 5.3 System Context Beispiel-Auswerte-SoC

Aufbauend auf der Analyse der Anforderungsfälle folgt als Teil der hier entwickelten Modellierungsmethode die Dekomposition der Funktionen. Hierzu werden zu Beginn aus den Anwendungsfällen Funktionen abgeleitet. In diesem Fall zeigt Abbildung 5.4 die abgeleiteten Funktionen „*readout-pressure-Sensor*“ als Allokation des Anwendungsfalls „*readout-pressure-Signal*“ und die Funktion „*readout_Temperature*“ als Allokation des Anwendungsfalls „*readout Temperature*“.

Anschließend wurden die Funktionen in die hier gezeigten Teilfunktionen dekomponiert. Die Modellierung kann dabei in einzelnen BDDs für einzelne Funktionen bzw. wie im vorliegenden Fall in Funktionsblöcken erfolgen oder es wird ein *Hierarchical Functional Diagram* für die gesamte Funktionalität des SoCs erstellt. Das hier gezeigte Diagramm bildet daher nur einen Teil der gesamten Funktionalität des Auswerte-SoCs ab.

Neben der Dekomposition der Funktionen sollte bei der Modellierung eine Beschreibung der Abläufe in Form von Activity Diagrams erfolgen. Dabei wird nach der in dieser Arbeit entwickelten Methode für jeden Block, welcher eine Funktion repräsentiert, ein Activity, welches dem Activity Diagram entspricht, modelliert. Abbildung 5.5 zeigt als Beispiel ein Activity Diagram als funktionalen Ablauf der „*readout-pressure-Sensor*“-Funktion des Auswerte-SoCs. Wie in der Abbildung 5.5 erkennbar ist, geht es dabei um eine sehr abstrakte Beschreibung der Abläufe. Diese dient in erster Linie dazu, die Funktionen des Auswerte-SoCs funktional in Beziehung zu setzen. Das Activity Diagram enthält somit alle in Abbildung 5.4 durch die Dekomposition erhaltenen Teilfunktionen der Funktion „*readout-pressure-Sensor*“.

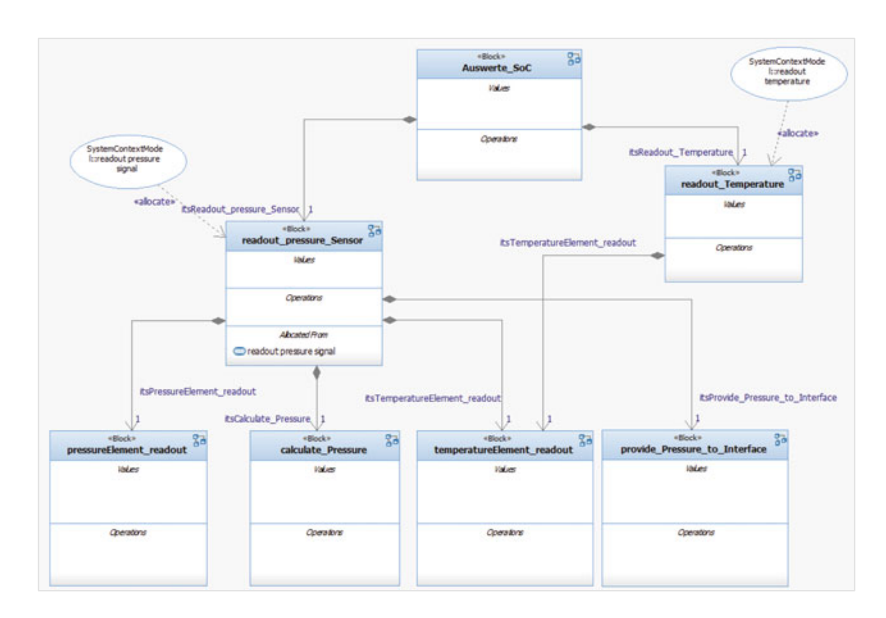


Abbildung 5.4 Hierarchical Functional Diagram Beispiel-Auswerte-SoC

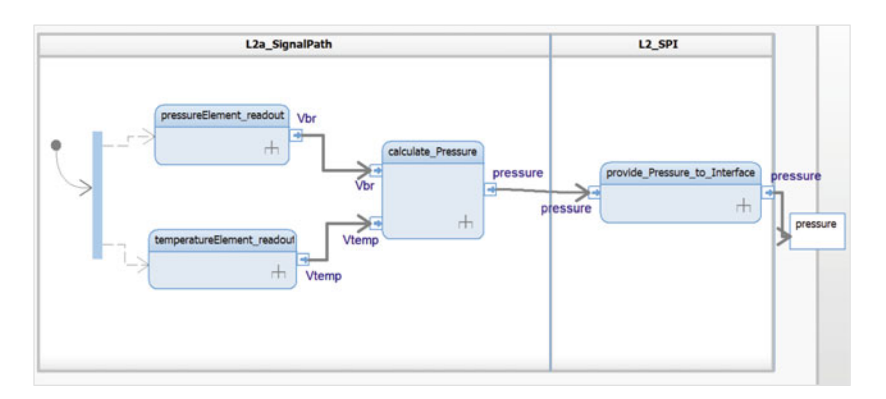


Abbildung 5.5 Activity Diagram Functional Model Beispiel-Auswerte-SoC

Die Abbildung 5.5 zeigt zudem die bereits in Abschnitt 4.1.2 behandelten *Swimlanes*, welche nach der hier angewendeten Modellierungsmethode für die Allokation der Funktionen zu den Modulen des *Technical Model* dienen. Hat die Dekomposition der Funktionalität einen gewissen Reifegrad erreicht, kann die Allokation der Funktionen aus dem *Functional Model* zu den Modulen des *Technical Model* erfolgen. Dabei findet zu Beginn meist noch keine Einteilung und Zuordnung der Funktionen in die jeweiligen Domänen statt. Die Modellierung der Dekomposition der Module sowie die Allokation erfolgt im *Hierarchical Technical Diagram*, wie in Abbildung 5.6 dargestellt.

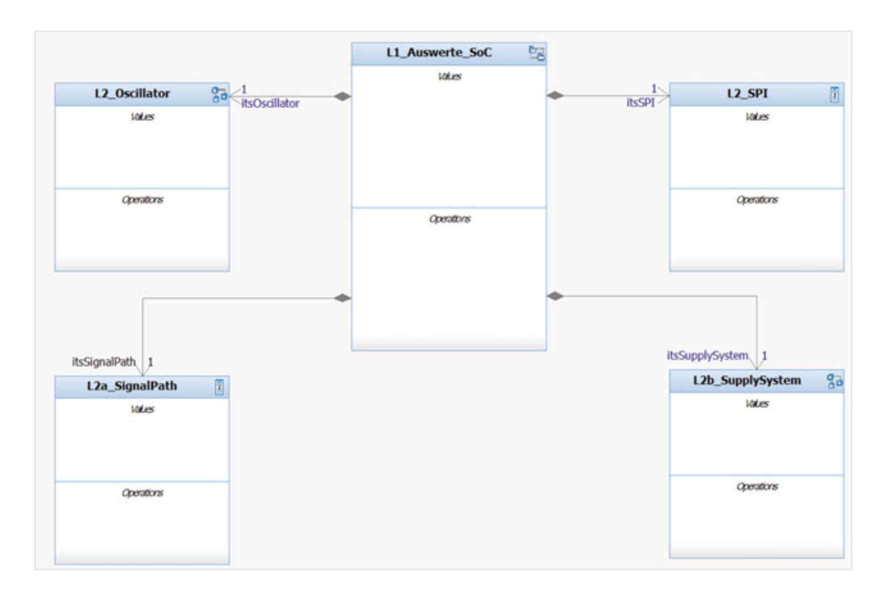


Abbildung 5.6 Hierarchical Technical Diagram Beispiel-Auswerte-SoC

Dabei bleiben die Entwicklung und Modellierung in der Regel auf der System-Ebene als Maß der Abstraktion. Anders als im *Hierarchical Functional Diagram* sollte im *Hierarchical Technical Diagram* zumindest jede Hierarchieebene der Dekomposition vollständig in einem Diagramm dargestellt werden. Lediglich um eine weitere Ebene der Dekomposition für eine der hier gezeigten Submodule vorzunehmen und zu modellieren, kann es sich bei sehr komplexen Systemen

anbieten, diese nächsttiefere Ebene als weiteres BDD hinter dem jeweiligen Block zu hinterlegen. Für die Modellierung der in Abbildung 5.6 dargestellten Module wurden die vorgefertigten Blöcke aus dem in dieser Arbeit entwickelten Automotive-SoC-Profil in das Model kopiert und für das entsprechende Modul angepasst.

Parallel zum *Hierarchical Technical Diagram* wurde das *Architecture Diagram,* wie in Abbildung 5.7 dargestellt, modelliert. Hierbei werden, nach der hier entwickelten Modellierungsmethode, die zu den Blöcken aus dem *Hierarchical Technical Diagram* zugehörigen Parts als eine Art Instanz des Blocks zur Darstellung des Moduls in der Architektur modelliert. Wird ein Teil der Funktionalität mittels Software auf einem Mikrochip realisiert, erfolgt die Modellierung der Software-Architektur in einem gesonderten IBD.

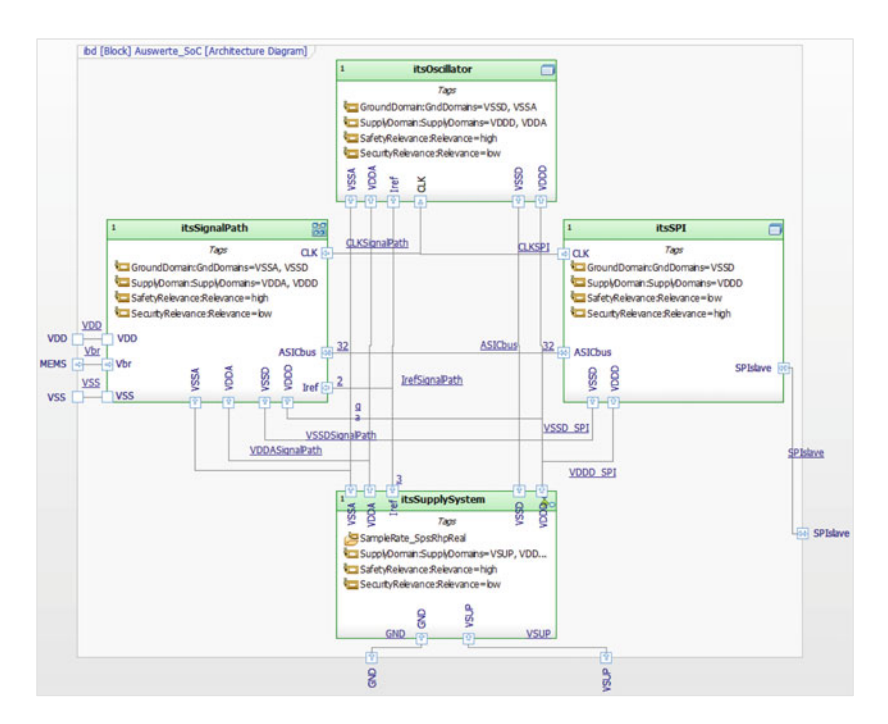


Abbildung 5.7 Architecture Diagram Beispiel-Auswerte-SoC

Vergleichbar zum *System Context Diagram* wurden als Teil des *Architecture Diagram* die Ports und Verbindungen der Architektur modelliert sowie die nichtfunktionalen Eigenschaften mittels Tags spezifiziert. Durch die Verwendung der vorgefertigten Blöcke aus dem Automotive-SoC-Profil für die Modellierung des *Hierarchical Functional Diagram* wurden ebenso vorgefertigte Parts für die Modellierung der Architektur erzeugt. Diese enthielten bereits eine große Menge der in der Abbildung 5.7 gezeigten Ports und Tags. In Abbildung 5.7 außerdem zu sehen ist die Modellierung der externen Ports des Auswerte-SoCs auf dem Diagramm-Rahmen des *Architecture Diagram.*

Nach der konzeptuellen Modellierung der Architektur von Hardware und gegebenenfalls Software endet in der Regel die Konzeptphase. Die Verfeinerung der Modellierung als Teil der Entwurfsphase wird in Abschnitt 5.1 behandelt.

5.1.3 Modellbasierte Spezifikation

Neben der Modellierung des Systems beinhaltet die in dieser Arbeit entwickelte Methode die bidirektionale Verlinkung der Systemanforderungen in natürlicher Sprache mit den Modellelementen des Systemkonzepts bzw. Systemmodells. Als essenzieller Bestandteil der Methode soll hierdurch eine kombinierte Spezifikation basierend auf zwei Spezifikationsdokumenten, welche in zwei verschiedenen Tools verwaltet werden, geschaffen werden. Aus diesem Grund soll nachfolgend die toolbasierte Lösung für die bidirektionale Verlinkung der Spezifikationsdokumente vorgestellt werden. Die dabei verwendete Tool-Umgebung wurde in Abschnitt 5.1.1 vorgestellt sowie deren Nutzen für die Methode dieser Arbeit behandelt.

Die Realisierung der Verlinkung erfolgt auf Basis der IBM-Rational-Tool-Umgebung, welche bereits die Verlinkung der Systemanforderungen, in DNG als Artefakte gespeichert, mit den Rhapsody Modellelementen ermöglicht. Somit sind durch die Auswahl der Tool-Umgebung für die hier vorliegende Arbeit die technischen Voraussetzungen für die Verlinkung gegeben. Die methodischen Voraussetzungen, um eine Spezifikation aufgeteilt auf zwei Tools mit unterschiedlichen Datenbasen zu schaffen, mussten in dieser Arbeit ermöglicht werden. Abbildung 5.8 veranschaulicht die Verlinkung der Kunden- bzw. Systemanforderungen in DNG mit den Modellelementen des Systemmodells in Rhapsody, ohne dabei auf die technische und Tool-spezifische Realisierung einzugehen.

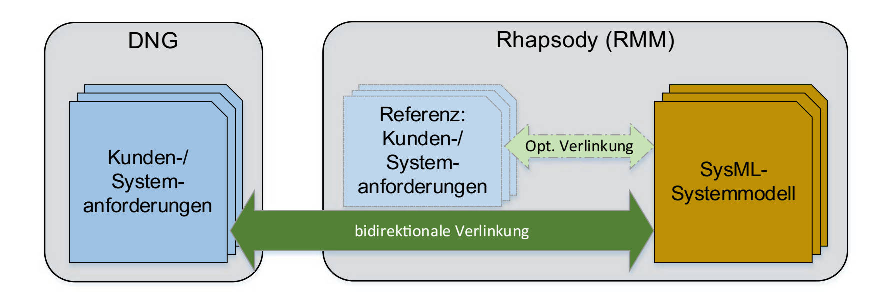


Abbildung 5.8 Verlinkung DNG und Rhapsody

Die grauen Blöcke der Tools zeigen gleichzeitig den sogenannten Master der jeweiligen Daten. Die Systemanforderungen sollten zur Sicherstellung der Konsistenz immer in DNG spezifiziert werden. Um die Verlinkung in Rhapsody zu ermöglichen, werden die beiden Datenbasen verbunden und so eine Referenz und damit eine Repräsentation der Systemanforderungen aus DNG in Rhapsody erreicht. Die Modelle aus Rhapsody werden dabei auf dem RMM gespeichert und mittels bidirektionaler OSLC-Verlinkung mit DNG verbunden. Die Verlinkung kann dabei sowohl in DNG wie auch in Rhapsody erfolgen.

Darüber hinaus lassen sich die Verlinkungen als Beziehungen in den Diagrammen darstellen. Diese Funktion wurde hier schematisch in der Abbildung 5.8 zur Veranschaulichung als „Optische Verlinkung" bezeichnet. Abbildung 5.9 zeigt einen Auszug des in Abbildung 5.4 gezeigten *Hierarchical Functional Diagram* und veranschaulicht dabei die Satisfy-Beziehung der *„readout-pressure-Sensor"*-Blocks zur hier beispielhaft dargestellten Kundenanforderung *Provide pressure measurment*.

Dabei wurde in diesem Fall die Repräsentation der Anforderung im Diagramm als Requirements-Block gewählt. Diese Darstellung kann schnell zu unübersichtlichen Diagrammen führen und sollte selektiv für den jeweiligen Fall gewählt werden. Alternativ ist es möglich, die Beziehung als Teil des Modells zu modellieren, ohne eine optische Repräsentation in einem Diagramm zu benötigen.

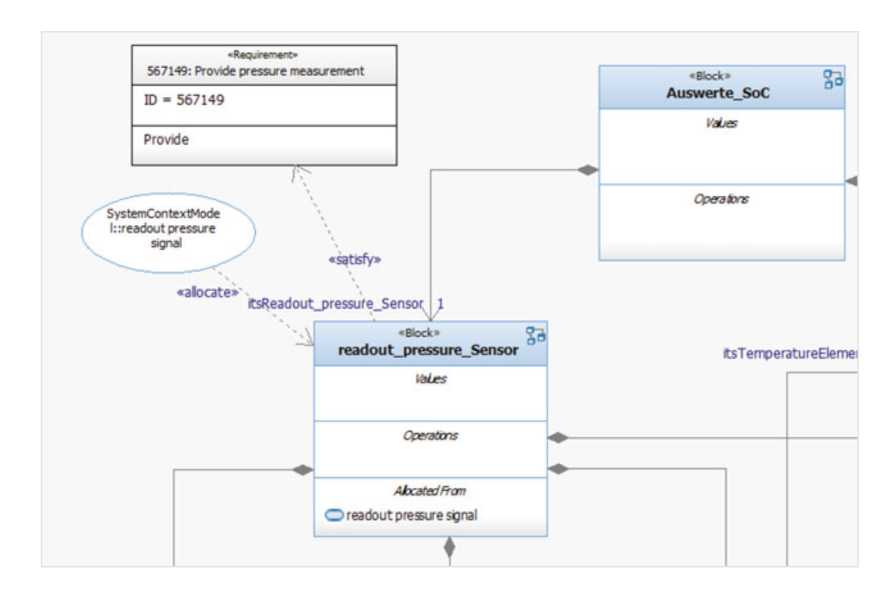


Abbildung 5.9 Beispiel: Optische Verlinkung Systemanforderungen in Rhapsody

5.1.4 Automatisierung der Modellierung

Neben der Nutzung der Plug-Ins für die Automatisierung des Entwurfs lassen sich Plug-Ins ebenso zur Automatisierung einzelner Arbeitsschritte bei der Modellierung nutzen. Hierdurch kann der Aufwand der Modellierung einer Spezifikation im Vergleich zur aktuellen Methode weiter gesenkt werden. So lässt sich über die bereits behandelte Java API von Rhapsody auch die Modellierung mittels Plug-Ins automatisieren. In dem in diesem Kapitel zugrunde liegenden industriellen Beispiel eines Auswerte-SoCs wurden beispielsweise spezifische Darstellungseigenschaften für „Stakeholder“-spezifische Sichten definiert.

Die Anwendung auf ein Diagramm würde jedoch in vielen Fällen eine manuelle Bearbeitung jedes einzelnen Modellelements eines Diagrammes benötigen. Die Einstellungsmöglichkeiten der Darstellung eines Modellelements in Rhapsody wird in den „Display Options“ (Anzeigeoptionen) zusammengefasst und kann dort mittels Dialogfenstern angepasst werden. Dies führt jedoch zu einem erheblichen manuellen Aufwand und birgt die Gefahr einer Abweichung von der definierten Standarddarstellung. Neben der Einstellung der „Display Options“

im Dialogfeld lassen sich viele dieser Einstellungen direkt mittels „Properties“ definieren.

In der hier vorliegenden Arbeit wurde daher ein Plug-In entwickelt, welches so weit wie möglich die Anpassung der „Display Options“ durch ein automatisiertes Setzen der „Properties“ ermöglicht. Dabei kann das Plug-In je nach „Stakeholder“-Sicht wiederverwendet und konfiguriert werden. Hierbei bietet Rhapsody die Option, das Starten und Ausführen des Plug-Ins direkt mittels Menüeintrag in Rhapsody zu ermöglichen. Zusätzlich zur Anpassung der „Properties“ bzw. der „Display Options“ schließt das Plug-In das aktuell geöffnete Diagramm auf, welches auf das Plug-In angewendet wurde und lädt es neu, um die neue Darstellung auf das Diagramm anzuwenden.

Um eine einheitliche Darstellung bei der Modellierung zu fördern und dabei den Aufwand so gering wie möglich zu halten, wurde als Erweiterung der Methode eine standardisierte Darstellung bei den IBDs definiert. Diese Darstellungsregeln lassen sich auch nachträglich mittels Plug-Ins auf die IBDs anwenden und werden dem Nutzer als Teil des Profils zur Verfügung gestellt. Es ist hierbei ebenfalls möglich, den Anwender/-innen eine Auswahl verschiedener Plug-Ins zur Verfügung zu stellen, welche ein bestimmtes Set aus Darstellungsregeln automatisiert anwendet. Die einheitliche Darstellung unterstützt den Leser des Modells dabei, den Sinn der Informationen leichter zu verstehen. Eine einheitliche Darstellung ist hierbei jedoch besonders wichtig, da im Laufe eines Entwicklungs-Flows eine Vielzahl verschiedener Personen an der Modellierung des Systems beteiligt sind.

5.2 Übergang Konzeptphase – Entwurfsphase

Im vorangegangenen Abschnitt wurde die Modellierung des Systemkonzeptes anhand eines Beispiel-SoCs veranschaulicht. Die Entwicklung findet hier in der Regel rein konzeptuell statt und endet daher in der Abstraktion auf der System-Ebene. Beim Übergang von der Konzept- zur Entwurfsphase wird, wie in Abschnitt 4.2 beschrieben, das modellierte Systemkonzept zum Systemmodell und damit zu einem Bestandteil der Entwurfsspezifikation. Um die in Abschnitt 4.3 beschriebene modellgetriebene Automatisierung des Systementwurfs zu erreichen, bedarf es einer Verfeinerung des Systemmodells. Nachfolgend wird diese Verfeinerung als Teil des in dieser Arbeit entwickelten Spezifikations-Flows an einem weiteren SoC-Beispiel behandelt und mittels vereinfachter Beispieldiagramme veranschaulicht. Dabei orientiert sich dieses Beispiel an einem SoC-Projekt in der Entwurfsphase.

5.2.1 Verfeinerung (Bus-)Architektur

Die Architekturbeschreibung ist bereits im *Technical Model* des Systemkonzepts in Form des *Architecture Diagram* enthalten und muss in der Entwurfsphase, wie in Abschnitt 4.3.2 beschrieben, lediglich verfeinert werden.

Die Modellierung und Generierung der Architektur wird nachfolgend am Beispiel des *Processor-Subsystems* veranschaulicht. Dieses beinhaltet die kontrollflussorientierten Module des Beispiel-SoCs. Für die Beschreibung der Architektur wird, wie in Abschnitt 4.1 beschreiben, im ersten Schritt die Dekomposition mittels BDD modelliert. Abbildung 5.10 zeigt das *Processor-Subsystem* mit den enthaltenen kontrollflussorientierten Modulen als Beispiel eines *Hierarchical Technical Diagram.*

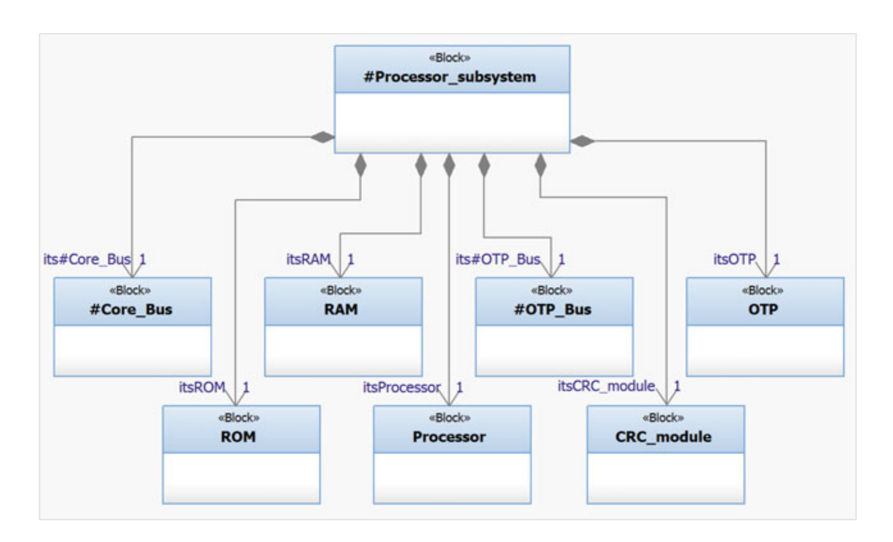


Abbildung 5.10 Hierarchical Technical Diagram Technical Model

Das *Processor-Subsystem* besteht in diesem Beispiel aus dem *Processor* als Repräsentation eines spezifischen Mikrocontrollers, dem zugehörigem *RAM* und *ROM*, zwei Bussen, dem *CRC Module* sowie dem „One-Time-Programmable" (*OTP*)-Modul. Aufgrund der geringen Relevanz der technischen Funktionalität der einzelnen Module für die hier vorliegende Arbeit, werden diese nicht näher behandelt.

Durch die Modellierung der Hardware-Submodule im BDD als *Directed Composition* des *Cortex-Subsystems* werden entsprechende Parts der Hardware-Submodule erzeugt. Diese Teile werden zur Modellierung des *Architecture Diagram* im IBD verwendet. Abbildung 5.11 zeigt das *Architecture Diagram* des *Processor-Subsystem* als IBD.

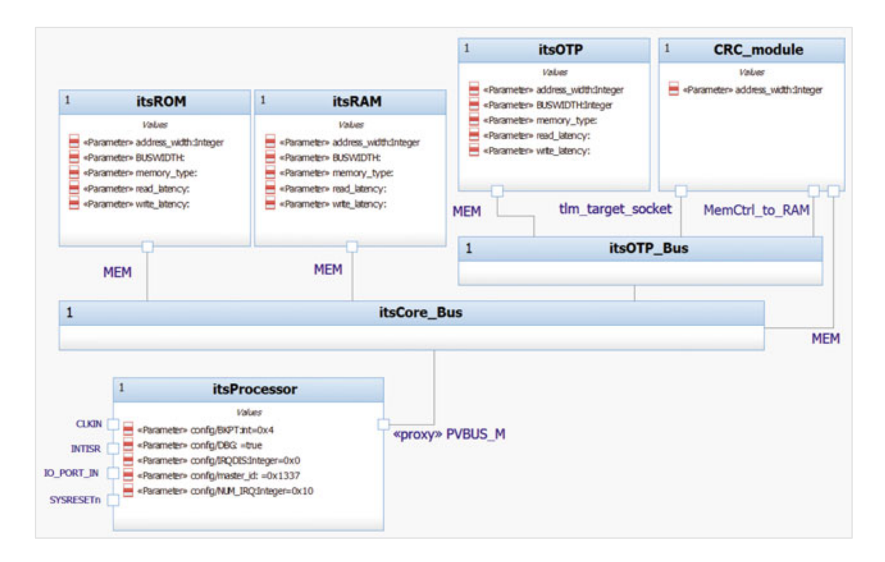


Abbildung 5.11 Architecture Diagram Technical Model Beispiel 1

Ein- und Ausgänge der Hardwaremodule werden im *Architecture Diagram* als sogenannte „ProxyPorts" modelliert. Darüber hinaus sind in der Abbildung 5.11 für die Module bereits Beispiel „ValueProperties" definiert, welche die Eigenschaften der Module spezifizieren und zudem der Generierung der Architekturbeschreibung in SystemC für den Virtuellen Prototyp dienen.

Als Teil der Verfeinerung der Architektur wird die Modellierung von On-Chip-Bussen durchgeführt. Wie in Abbildung 5.11 und Abbildung 5.10 zu sehen, werden die Busse im *Architecture Diagram*, wie alle Module der Architektur, als Blöcke im BDD und als Parts im IBD modelliert. In der Entwurfsphase reicht diese abstrahierte Darstellung der Busse meist jedoch nicht mehr aus und es wird zusätzlich zur Architektur des SoCs auf System-Ebene eine Modellierung der Busarchitektur benötigt. Abbildung 5.12 zeigt die modellierte Busarchitektur des in diesem Kapitel genutzten Beispiel-SoCs.

Dabei wird zur Modellierung der nächsttieferen Abstraktionsebene in diesem Beispiel hinter dem Part *itsCore-Bus* ein weiteres IBD mit der Busarchitektur hinterlegt. Vergleichbar mit dem *Architecture Diagram* werden hierzu die Module, welche über den Bus verbunden sind, als Parts modelliert sowie die Verbindungen zwischen den Modulen, welche Bestandteil des Busses sind, als *Connectoren* abgebildet.

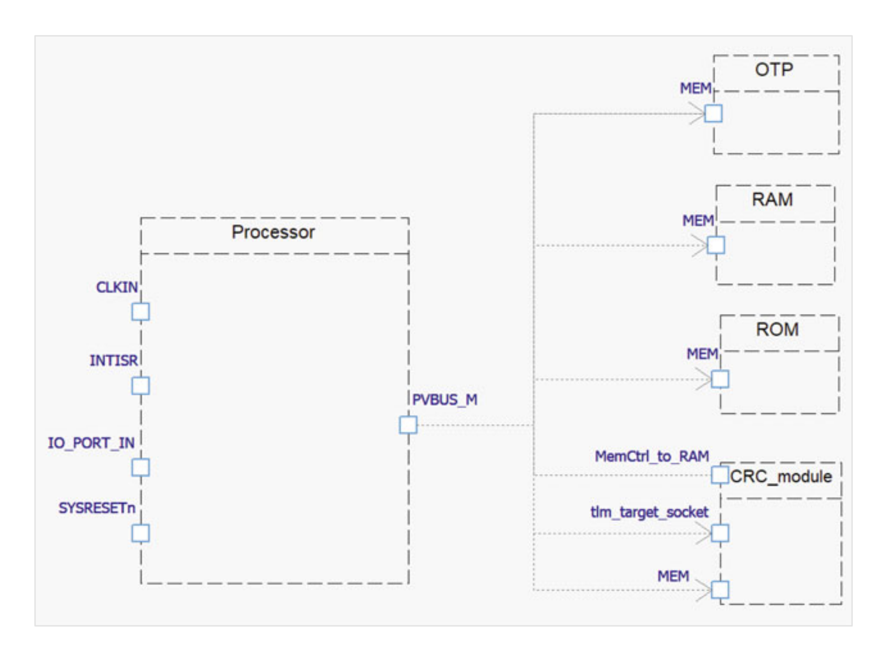


Abbildung 5.12 Busarchitektur Technical Model

Für die Implementierung der nachfolgend in Abschnitt 5.3.4 beschriebenen modellgetriebenen Verifikation wurde die Architektur des SoCs aus Abbildung 5.11 zusätzlich verfeinert, wie in Abbildung 5.13 gezeigt. Dabei wurde dem Modell das *CRC-OTP-Subsystem* hinzugefügt und die Architektur der bereits in Abbildung 5.10 enthaltenen Module *OTP* und *CRC Module* wie folgt modelliert bzw. verfeinert.

Da in dieser Arbeit der Fokus auf der Vorstellung der Methode zur modellbasierten Systementwicklung und dem modellgetriebenen Entwurf liegt, wird in diesem Abschnitt nicht näher auf die Funktionen der einzelnen Submodule in dem hier gezeigten Beispiel-SoC eingegangen. Abbildung 5.13 zeigt beispielhaft die

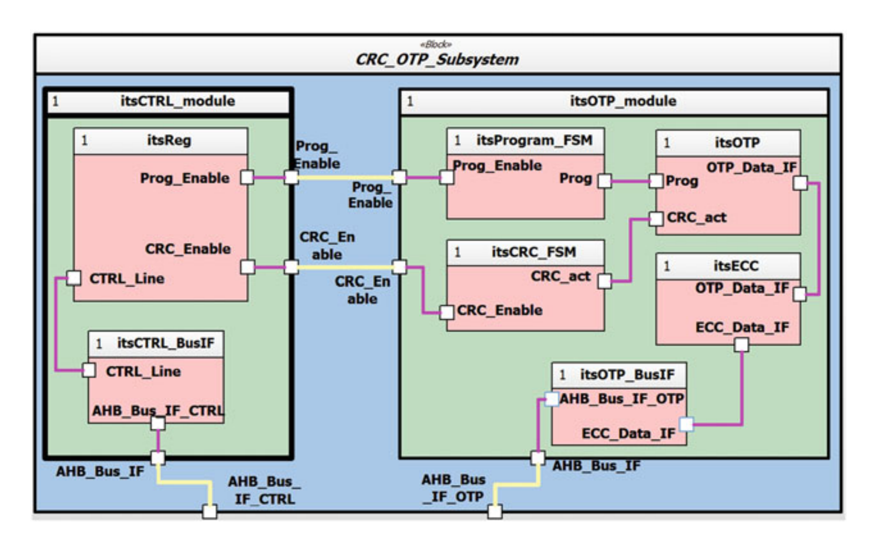


Abbildung 5.13 Architecture Diagram Technical Model Beispiel 2

Möglichkeit einer detaillierten und implementierungsnahen Modellierung, welche für die modellgetriebene Automatisierung in der Regel benötigt wird.

5.2.2 Verfeinerung der Verhaltensbeschreibung

Neben der Verfeinerung der Architektur erfolgt, wie in Abschnitt 4.2.2 beschrieben, in der Entwurfsphase eine Spezifikation und Modellierung des Modulverhaltens als Teil des *Behavioral Model*. Neben der eindeutigen Spezifikation wird durch die Formalisierung und Modellierung des Verhaltens ein implementierungsnahes und maschinenlesbares Format erreicht und somit der Grundstein für die Realisierung eines modellgetriebenen Systementwurfs gelegt. Die dabei entstehenden Flowchart Diagrams sind im *Behavioral Model* in ein Hardware- und ein Software-Package unterteilt. Hierdurch wird eine Generierung verschiedener Programmiersprachen für das jeweilige Package des SoC-Modells ermöglicht. Für die Definition der Zielsprache wird für einen definierten Teil des Modells, wird im Modell eine Rhapsody-spezifische Konfigurationsdatei angelegt. Auf diese Weise ist es möglich, C++-Code bzw. SystemC-Code lediglich für den Teil der verhaltensbeschreibenden Diagramme zu generieren, welcher für die

Generierung des Virtuellen Prototyps, wie in Abschnitt 4.3.1 beschrieben, benötigt wird. Da bei der Modellierung und Generierung der Verhaltensbeschreibung die Verwendung der korrekten Registernamen entscheidend ist, wurden mittels des in Abschnitt 4.1.4 beschriebenen automatisierten Transfers die Registernamen aus der IP-XACT-Registerbeschreibung in das SysML-Modell übertragen. Abbildung 5.14 zeigt einen Auszug der so übertragenen Registernamen, welche im Modell als *ValueProperties* vertreten werden und in der Modellierung der Verhaltensbeschreibung wie Variablen verwendet werden können.

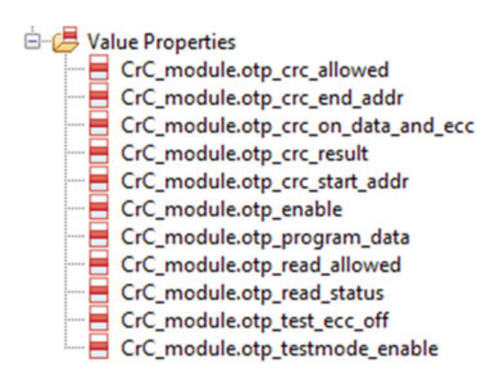


Abbildung 5.14 Auszug automatisiert übertragene Register-„ValueProperties“

Nachfolgend soll die Modellierung und anschließende Generierung der Hardware-Verhaltensbeschreibung anhand einer Teilfunktion des hier zugrunde liegenden Beispiel-SoCs veranschaulicht werden. Abbildung 5.15 zeigt hierzu das Flowchart Diagram einer zyklischen Redundanzprüfung (CRC-Berechnung) für einen 32 Bit-unsigned-Integer-Wert in Form einer modellierten For-Schleife.

Der Algorithmus der zyklischen Redundanzprüfung wird häufig zur Erkennung von Fehlern bei der Datenübertragung verwendet [90]. Hierzu wird der Wert der einzelnen Bits der Variablen „*input*“ betrachtet und an die CRC-Berechnungsfunktion übergeben. In der Abbildung 5.15 ist die stark implementierungsnahe Modellierung des Modulverhaltens zu erkennen, welche als Grundlage für die nachfolgend beschriebenen Lösungen der modellgetriebenen Automatisierung des Systementwurfes benötigt wird.

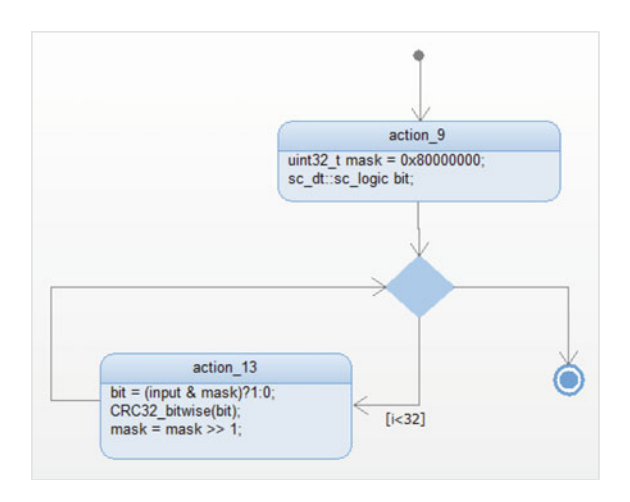


Abbildung 5.15 Flowchart Diagram Behavioral Model

5.3 Modellgetriebener Systementwurf und Verifikation

Nachfolgend wird das in dieser Arbeit durchgeführte „Proof-of-Concept" der in Kapitel 4 konzeptuell beschriebenen Methoden zur Steigerung der Effizienz bei der Implementierung des Systementwurfs sowie der Effizienz bei der Erstellung der Verifikation vorgestellt. Als Teil dessen wird die dabei verwendete Tool-Umgebung beschrieben sowie die in dieser Arbeit entwickelten konfigurierten Codegeneratoren für die Automatisierung des Entwurfs vorgestellt. Hierbei soll gezeigt werden, wie auf Grundlage der in Abschnitt 5.2 vorgestellten modellierten Teile der Entwurfsspezifikation die Codegenerierung ermöglicht werden kann.

5.3.1 Definition Tool-Umgebung

Synopsys® Virtualizer Studio (Virtualizer Studio) ist ein Eclipse-basiertes Tool, welches die Funktionalitäten mehrerer anderer Synopsys-Werkzeuge vereint und diese um eine eigene Anwenderoberfläche erweitert. Für die hier vorliegende Arbeit wurde Virtualizer Studio für die Instanziierung und Vernetzung von Hardware-IPs verwendet, um daraus Systeme oder Teilsysteme für die Implementierung eines Virtuellen Prototyps zu erstellen. Darüber hinaus lassen sich die Virtuellen Prototypen in Virtualizer Studio zusammen mit der Software

simulieren und testen. An vielen Stellen wird den Anwender/-innen zudem die Möglichkeit der Abstrahierung geboten, um so die Entwicklungszeiten und den Aufwand für die Implementierung des Virtuellen Prototyps zu verringern. Ein Vorteil bei der Verwendung von Virtualizer Studio ist das zur Verfügung stehende Python-„Application Programming Interface“ (API, Programmierschnittstelle) des Tools. Nahezu alle manuell ausführbaren Aktionen können hierdurch via Python-Skript automatisiert werden. Somit wird eine Möglichkeit der Automatisierung bei der Erstellung von Virtuellen Prototypen eröffnet und findet in dieser Arbeit wie im nachfolgenden Abschnitt 5.3.2 beschrieben Anwendung [70] [91].

5.3.2 Automatisierung der Entwicklung Virtueller Prototypen

Der folgende Abschnitt zur Automatisierung im Bereich der Virtuellen Prototypen baut auf der Arbeit aus [15] auf. Dieser Ansatz soll im ersten Schritt um die Generierung einer Verhaltensbeschreibung der kontrollflussorientierten Module erweitert werden, wie in Abschnitt 4.3 beschrieben und in Abbildung 4.12 dargestellt. Die Generierung wird dabei anhand des in Abschnitt 5.2 vorgestellten Modells eines Beispiel-SoCs behandelt.

Generierung der SystemC-Verhaltensbeschreibung

Um eine modellbasierte Generierung zu ermöglichen, muss zu Beginn der Entwurfsphase das Systemmodell, wie in Abschnitt 5.2.2 veranschaulicht, um das *Behavioral Model* und damit um die Verhaltensbeschreibung der einzelnen Hardwaremodule erweitert werden. Um die Generierung der SystemC-Verhaltensbeschreibung, wie in Abschnitt 4.3.1 vorgestellt, auf Basis des so erhaltenen *Behavioral Model* zu implementieren, wurde der in Rhapsody enthaltene C++-Generator für den hier vorliegenden Anwendungsfall konfiguriert. Die Generierung der SystemC-Verhaltensbeschreibung erfolgt dabei in zwei Schritten, wie in Abbildung 5.16 dargestellt.

Im ersten Schritt erfolgt eine Transformation des SysML-Diagram bzw. des Flowchart Diagram in ein *Simplified Model*. Dabei werden alle grafischen Modellelemente vereinfacht und auf die relevanten Metadaten für die Codegenerierung reduziert. Dieses *Simplified Model* ist eine reine Metadatenbeschreibung der Modellteile für die Codegenerierung und beinhaltet daher keine grafische Repräsentation. Im zweiten Schritt wird auf Basis des *Simplified Model* der SystemC-Code für die Hardware-Verhaltensbeschreibung generiert. Dabei lässt sich das Ergebnis der Generierung maßgeblich an drei Stellen des Flows beeinflussen: bei der Art der

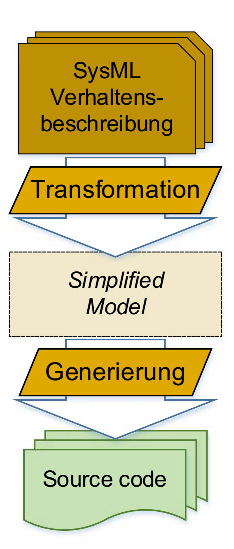


Abbildung 5.16 Modellgetriebene Generierung

Modellierung, bereits in Abschnitt 4.3 behandelt, bei der Transformation zwischen SysML-Modell und *Simplified Model* sowie bei der Generierung des Codes auf Basis des *Simplified Model*.

Das Ergebnis der auf Basis des in Abbildung 5.15 gezeigten Flowchart Diagram erfolgten Generierung wird in Abbildung 5.17 präsentiert. Zu beachten ist hierbei die Fensterung einzelner Code-Blöcke durch Kommentare, wodurch diese den ihnen zugehörigen Action-Blöcken im Modell zugeordnet werden können. Die Kommentare werden hierbei automatisch generiert und können durch die Anwender/-innen konfiguriert werden. Die Zählervariablen der For-Schleife im Code werden in der Decision Node des Flowchart Diagram definiert.

Wurde der SystemC-Code erfolgreich generiert, muss auch hier die generierte Verhaltensbeschreibung mit dem SystemC-Wrapper aus IP-XACT kombiniert werden. Als Ergebnis der Generierung aus Rhapsody entsteht für jedes Hardwaremodul eine Quelldatei und eine zugehörige Header-Datei. Diese Dateien entsprechen der Verhaltensbeschreibung einer Komponente für den Virtuellen Prototyp. Der generierte Wrapper wird in einer weiteren Header-Datei beschrieben. Das Zusammenführen des Wrappers und der zugehörigen Verhaltensbeschreibung funktioniert durch Vererbung der Header-Datei des Wrappers in die Header-Datei der Verhaltensbeschreibung. Dadurch werden die definierten Schnittstellen bekannt gemacht und

```
void CRC_module::CRC32_add_word(uint32_t input) {
    //#[ state CRC32_add_word(uint32_t).action_9.(Entry)
    uint32_t mask = 0x80000000;
    sc_dt::sc_logic bit;
    //#]
    //#[ transition CRC32_add_word(uint32_t).3
    //#]
    for (int i=0; i<32; i++)
    {
        //#[ state CRC32_add_word(uint32_t).action_13.(Entry)
                bit = (input & mask)?1:0;
                CRC32_bitwise(bit);
                mask = mask >> 1;
        //#]
    }

}
```

Abbildung 5.17 Beispiel Generierungsergebnis Flowchart Diagram (zu Abbildung 5.15)

eine vollständige Darstellung des Hardwaremoduls im Virtuellen Prototyp ermöglicht. Für die Integration und Konstruktion der generierten Module in den VP ist ein Import der Dateien in den Virtualizer erforderlich.

Generierung der SystemC-Architektur

Um den letzten Schritt der Implementierung der Virtuellen Prototypen ebenfalls zu automatisieren, wurde für die Generierung der SystemC-Architekturbeschreibung ein zweistufiger Transformations-Flow entwickelt. Die Generierung der Architektur erfolgt dabei auf Grundlage des in Abschnitt 5.2.1 vorgestellten *Architecture Diagram* (Abbildung 5.11) sowie dem Diagramm der Busarchitektur (Abbildung 5.12) des SoC-Beispiels. Dabei erfolgt die Generierung der SystemC-Architekturbeschreibung in zwei Schritten.

Im ersten Schritt werden alle benötigten Informationen der Architekturbeschreibung aus dem *Technical Model* mittels Plug-Ins in drei Tabellen übertragen. Diese Tabellen werden als CSV-Tabellen ausgegeben. Dabei werden in der ersten Tabelle die Instanzen der Komponenten eingetragen, in der zweiten Tabelle die Verbindungen zwischen den Komponenten untereinander und in der dritten Tabelle die Spezifikation in Form von Parametern der Komponenten. Tabelle 5.1 zeigt ein leicht abstrahiertes Beispiel einer CSV-Tabelle für die Instanzen. Sie enthält die beispielhaften Daten zweier Komponenten. Dabei handelt es sich um eine externe Komponente für die der Hersteller eine entsprechende IP-Bibliothek mitliefert und eine selbst entwickelte Komponente für welche die IP generiert wurde.

Tabelle 5.1 Beispiel CSV-Tabelle für die Architekturbeschreibungs-Generierung

InstanceName	**Vendor**	**IPLib**	**IPName**	**IPLibPath**
Processor	External Vendor	ARM_CORTEX_Example_Lib	ARM_CORTEX_Example	<PathToExternalXMLFile>
CRC_module	Internal Vendor	CRC_module_Lib	CRC_module	<PathToCustomXMLFile>

In der ersten Spalte (*Instance Name*) der Tabelle 5.1 befindet sich der Name der Instanz. Dieser entspricht dem Namen der Komponente im Modell. In der zweiten Spalte (*Vendor*) wird der Hersteller der IP genannt, da im Virtuellen Prototyp auch fertige IPs anderer Hersteller verbaut werden können. Spalte drei (*IPLIb*) beinhaltet den Namen der IP-Bibliothek, Spalte vier (*IPName*) den Namen der zu erzeugenden IP und Spalte fünf (*IPLibPath*) den Pfad, an dem die IP-Bibliothek gespeichert ist. Die Bibliotheken der IPs entsprechen den generierten SystemC-Wrappern, kombiniert mit der SystemC-Verhaltensbeschreibung des jeweiligen Hardwaremoduls.

Im zweiten Schritt werden die exportierten CSV-Tabellen, wie in Abbildung 5.18 veranschaulicht, mittels Python-Skript ausgelesen.

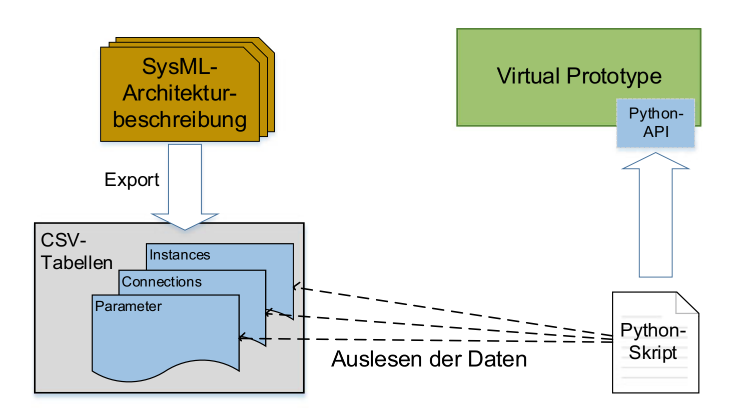


Abbildung 5.18 Architekturgenerierung Virtueller Prototyp

Anschließend werden die extrahierten Informationen über eine Python-API in den VP übertragen. Im Virtuellen Prototyp werden die Informationen entsprechend aufgelöst, es wird auf die IP-Bibliotheken zugegriffen und die IPs sowie die Architektur als Ganzes werden erstellt. Durch die hier gezeigte Erweiterung der teilautomatisierten Methode um den modellgetriebenen Ansatz für die kontrollflussorientierten Module lässt sich ein vollautomatisierter Generierungs-Flow für die Entwicklung von Hardware-VPs ermöglichen.

5.3.3 Automatisierung des Softwareentwurfs

Um zusätzlich zur Automatisierung der Entwicklung von Virtuellen Prototypen Teile des SoC-Entwurfs selbst zu automatisieren, wird nachfolgend die in Abschnitt 4.3 konzeptuell behandelte Generierung von Softwarecode auf Basis des SysML-Modells beschrieben. Dabei wurde ein vergleichbarer Ansatz zur Generierung der SystemC-Verhaltensbeschreibung gewählt.

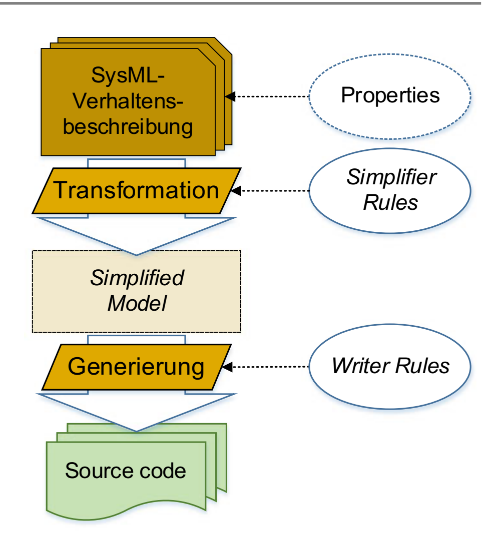
Die Verhaltensbeschreibung der Software findet mittels Flowchart Diagram als Teil des *Behavioral Model* statt. Dabei werden im *Behavioral Model* die Software-beschreibenden Flowchart Diagrams in ein separates Package eingeordnet, um für dieses Package eine C-Codegenerierung zu ermöglichen. Aufgrund der Ähnlichkeit der Modellierung zwischen Verhalten des Virtuellen Prototyps und der Software wurde auch für die Codegenerierung ein vergleichbarer Ansatz verfolgt. So wurde für die Generierung von Softwarecode auf Basis des SysML-Modells der in Rhapsody enthaltene C-Codegenerator für die hier vorliegende Arbeit entsprechend dem Anwendungsfall konfiguriert. Die Generierung erfolgt dabei ebenfalls, wie in Abbildung 5.16 für die Generierung der SystemC-Verhaltensbeschreibung dargestellt, in zwei Schritten.

- Im ersten Schritt erfolgt eine Transformation des SysML-Diagramms bzw. des Flowchart Diagram in ein *Simplified Model*.
- Im zweiten Schritt wird auf Basis des *Simplified Model* der SystemC-Code für die Hardware-Verhaltensbeschreibung generiert.

Dabei lässt sich das Ergebnis der Generierung, wie im vorangegangenen Abschnitt beschrieben, an drei Stellen des Flows beeinflussen. Diese sollen nachfolgend näher behandelt werden und gelten gleichermaßen für die Generierung der SystemC-Verhaltensbeschreibung aus dem vorangegangenen Abschnitt. Abbildung 5.19 veranschaulicht, aufbauend auf Abbildung 5.16, die drei Möglichkeiten der Einflussnahme auf die Codegenerierung.

- **Properties**: Wie in Abschnitt 5.1.1 beschrieben, verfügt Rhapsody über „Properties“ zur Anpassung der Modellierung, der Nutzeroberfläche sowie zur Anpassung der Codegenerierung. Der Einfluss der „Properties“ auf die Generierung greift im ersten Schritt der Generierung bei der Transformation des SysML-Modells in ein *Simplified Model*. So lassen sich beispielsweise mithilfe der „Properties“ die Generierung von Setter- und Getter-Funktionen für die Variablen aktivieren oder deaktivieren.

Abbildung 5.19
Einflussnahme Codegenerierung



- **Simplifier Rules**: Mithilfe von sogenannten *Simplifier Rules* lassen sich SysML-Modelle zu *Simplified Models* transformieren. Dabei dienen die Simplifier Rules als Regeln für die Transformation und können von den Anwendern/-innen konfiguriert werden. Diese *Simplifier Rules* können in Form von Plug-Ins zur Verfügung gestellt werden oder es können bestehende Regeln manuell erweitert werden. Über „Properties" lässt sich in Rhapsody definieren, ob die Standardregeln bei der Transformation angewendet werden oder nutzerspezifische Regeln zum Einsatz kommen sollen.
- **Writer Rules**: Das Regelwerk für die Generierung des Codes auf Basis des *Simplified Model* wird über sogenannte *Writer Rules* definiert. Dieses Regelwerk kann ebenfalls erweitert oder bestehende Regeln durch eigene ersetzt werden.

5.3.4 Modellgetriebene Verifikation

In diesem Abschnitt wird die in dieser Arbeit entwickelte Generierung im Bereich der Verifikationserstellung vorgestellt. Dazu soll anhand der in Abbildung 5.13 gezeigten Architektur des *CRC_OTP_Subsystems* die Generierung von „Connectivity Checks", wie in Abschnitt 4.3.3 konzeptuell beschrieben, veranschaulicht werden.

In dem in Abbildung 5.13 gezeigten *Architecture Diagram* sind die Ports aller Hierarchieebenen abgebildet und mittels *Connectoren* verbunden. Dabei sind einzelne Ports direkt innerhalb eines Moduls miteinander verbunden und andere Verbindungen gehen über mehrere Hierarchieebenen hinweg. Für die Generierung des „Connectivity Checks“ wird im ersten Schritt zu jedem Anfangs-Port der End-Port im *Architecture Diagram* mittels Java-Skript gesucht und anschließend in eine Tabelle übertragen. Liegen, wie in Abbildung 5.13 gezeigt, bei der Verbindung zwischen Port *Prog_Enable* des Moduls *itsReg* und Port *Prog_Enable* des Moduls *itsProgram_FSM* weitere Ports, sollen diese nicht in die Tabelle eingetragen werden.

Um diese Information in die Tabelle übertragen zu können, wird ein weiteres Skript benötigt. Im Modell verfügt jedes Modellelement über eine eigne ID. Diese ID des End-Ports wird in einem Tag automatisiert zwischengespeichert, um im nächsten Schritt den Namen des zugehörigen Parts sowie dessen hierarchische Ordnung im Modell mittels der ID des End-Ports zu extrahieren und in die Tabelle einzutragen.

Die Informationen der so erhaltenen Tabelle werden im nächsten Schritt mittels Python-Skript [92] ausgelesen und auf Grundlage der so erhaltenen Port-Variablen SystemVerilog-Assertions generiert. Die Assertions prüft hierbei lediglich, ob der Anfangs-Port *portX* mit dem End-Port *portY* im Entwurf verbunden ist. Abbildung 5.20 zeigt einen Auszug der auf Basis des in Abbildung 5.13 gezeigten *Architecture Diagram* generierten SystemVerilog-Assertions.

```
always_comb a_sig_2: assert final (
  crc_otp_subsystem.ctrl_module_i.reg_i.crc_enable === crc_otp_subsystem.otp_module_i.crc_fsm_i.crc_enable)
  else $error("Connection from 'crc_otp_subsystem.ctrl_module_i.reg_i.crc_enable' to
  'crc_otp_subsystem.otp_module_i.crc_fsm_i.crc_enable' is broken");

always_comb a_sig_3: assert final (
  crc_otp_subsystem.ctrl_module_i.reg_i.ctrl_line === crc_otp_subsystem.ctrl_module_i.ctrl_busif_i.ctrl_line)
  else $error("Connection from 'crc_otp_subsystem.ctrl_module_i.reg.ctrl_line' to
  'crc_otp_subsystem.ctrl_module_i.ctrl_ctrl_busif_i.ctrl_line' is broken");

always_comb a_sig_4: assert final (
  crc_otp_subsystem.ctrl_module_i.reg_i.prog_enable === crc_otp_subsystem.otp_module_i.program_fsm_i.prog_enable)
  else $error("Connection from 'crc_otp_subsystem.ctrl_module_i.reg_i.prog_enable' to
  'crc_otp_subsystem.otp_module_i.program_fsm_i.prog_enable' is broken");
```

Abbildung 5.20 Auszug Ergebnis Generierung Assertions

In der Abbildung 5.20 ist unter anderem das Ergebnis der Generierung für das besprochene Beispiel der Verbindung zwischen Port *Prog_Enable* des Moduls *itsReg* und Port *Prog_Enable* des Moduls *itsProgram_FSM* zu sehen. Die Automatisierung der Entwicklung von „Connectivity Checks“ auf Basis des SysML-Modells zeigt die Möglichkeiten der Effizienzsteigerung im Bereich der Verifikation auf. Durch den modellgetriebenen Ansatz lässt sich in diesem Gebiet

nicht nur der Aufwand durch die Qualitätssteigerung der Spezifikation erhöhen, es lässt sich zudem der Aufwand bei der Entwicklung der Verifikation reduzieren.

5.4 Zwischenergebnis

In Kapitel 5 wurde die Implementierung der zuvor in Kapitel 4 beschriebenen modellbasierten Entwicklungsmethode beschrieben. Dabei wurde anhand eines SoC-Beispiels der in dieser Arbeit entwickelte Spezifikations-Flow durchlaufen und mittels Beispiel-Diagrammen veranschaulicht. Darüber hinaus wurde auf Basis eines zweiten Beispiel-SoCs, welcher den Entwicklungsstand des SoCs in der Entwurfsphase beispielhaft repräsentiert, die Automatisierung des Systementwurfs und der Verifikationserstellung erläutert. Als Teil dessen wurde die Implementierung der in dieser Arbeit genutzten und entwickelten Generatoren für die modellgetriebene Generierung vorgestellt. Die Evaluierung der modellbasierten Entwicklungsmethode und die damit verbundenen Lösungen für die Automatisierung folgt in Kapitel 6.



6 Evaluierung

Die in dieser Arbeit entwickelte modellbasierte Entwicklungsmethode wurde mittels zweier verschiedener Techniken evaluiert. Zum einen fand die Evaluierung des Einflusses der modellbasierten Entwicklungsmethode auf die SoC-Entwicklung, wie nachfolgend in Abschnitt 6.1 beschrieben, durch Interviews statt. Zum anderen wurde der als Ergebnis der modellgetriebenen Automatisierung erhaltene Code auf Korrektheit und Performance-Eigenschaften mittels Co-Simulation getestet und mit dem manuell implementierten Code verglichen. Das Vorgehen sowie das Ergebnis dieser Tests wird in Abschnitt 6.2 vorgestellt.

6.1 Interviewevaluierung

Der Einfluss der in dieser Arbeit entwickelten Methode auf die SoC-Entwicklung erfolgte mittels einer zweistufigen Interviewevaluierung. Dabei orientiert sich das hier gezeigte Vorgehen an der Methode für die qualitative Inhaltsanalyse nach Mayring [93]. Da jedoch die Evaluierung der in dieser Arbeit entwickelten modellbasierten Entwicklungsmethode im Mittelpunkt der Befragung steht, weicht das Vorgehen in einigen Punkten vom Standard ab. So wurde für die Interviewevaluierung ein strengerer Rahmen in Form eines Leitfadens gewählt, um den Fokus der Befragung auf das entsprechende für diese Arbeit relevante Themengebiet zu lenken [17].

Im diesem Kapitel wird zu Beginn in Abschnitt 6.1.1 das Vorgehen für die Datenerhebung beschrieben. Auf die Technik bei der Analyse der erhobenen Daten wird in Abschnitt 6.1.2 näher eingegangen. Die Beschreibung der Auswahl der Interview-Teilnehmer/-innen erfolgt in Abschnitt 6.1.3. Am Ende des Kapitels folgen in Abschnitt 6.1.4 die Vorstellung sowie eine Diskussion der Ergebnisse aus der Interviewevaluierung.

A. Kirchner, *Entwicklung von Methoden zur abstrakten Modellierung von Automotive Systems-on-Chips*, https://doi.org/10.1007/978-3-658-38437-1_6

6.1.1 Datenerhebung

Für die Datenerhebung wurden Interview-Teilnehmer/-innen der Firma *Robert Bosch GmbH* in Reutlingen aus dem Bereich der SoC-Entwicklung mittels zweier unterschiedlicher, jedoch aufeinander aufbauender Interviewmethoden befragt. Dabei wurde folgendes Vorgehen für die Datenerhebung verwendet (Abbildung 6.1 veranschaulicht ergänzend das Vorgehen der Datenerhebung):

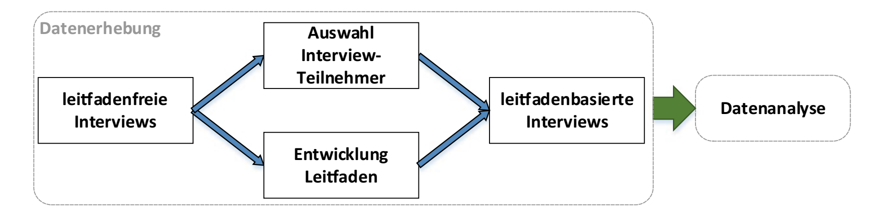


Abbildung 6.1 Datenerhebung Interviewevaluierung

- Im ersten Schritt wurden leitfadenfreie Interviews mit Entwicklern/-innen verschiedener Fachbereiche aus der SoC-Entwicklung über aus ihrer Sicht aktuell bestehende Defizite in der SoC-Entwicklung durchgeführt. Dazu wurde ein Fragenkatalog für die Befragung der Teilnehmer/-innen definiert, ohne dabei jedoch Antwortmöglichkeiten als Leitfaden vorzugeben. Die Teilnehmer/-innen sollten frei über die aus ihrer Sicht wichtigsten Defizite in der SoC-Entwicklung berichten.
- Darauf aufbauend wurden aus den leitfadenfreien Interviews fünf Defizite ausgewählt, welche von mehreren Teilnehmern/-innen der leitfadenfreien Interviews genannt wurden und sich aus deren Sicht stark auf die Effizienz der SoC-Entwicklung auswirken.
- Auf Grundlage dieser fünf ausgewählten Defizite und den erhaltenen Informationen aus den leitfadenfreien Interviews wurde ein Leitfaden für die leitfadenbasierten Interviews und damit die Interviewevaluierung erarbeitet.
- In der zweiten Stufe der Interviewevaluierung wurden Teilnehmer/-innen ausgewählt, welche bereits mehrjährige Erfahrungen in der SoC-Entwicklung vorweisen sowie bereits an der Entwicklung eines Pilotprojekts nach der neuen modellbasierten Entwicklungsmethode beteiligt waren.

- Bei diesen Interviews wurden den Teilnehmer/-innen im Leitfaden definierte Fragen unterschiedlicher Themenschwerpunkte gestellt, bei deren Beantwortung zudem eine vordefinierte Anzahl an Antworten zu Verfügung stand. Die Fragen sowie die Antwortwortmöglichkeiten der leitfadenbasierten Interviews werden in Abschnitt 6.1.4 näher behandelt.

Auf der Grundlage der erhobenen Daten erfolgte deren Analyse und Quantifizierung, wie im nachfolgenden Abschnitt beschrieben.

6.1.2 Datenanalyse der leitfadenbasierten Interviews

Die Fragenkomplexe 1–3 sollten dazu dienen, auf Grundlage der Einschätzungen bzw. Antworten der Interview-Teilnehmer/-innen Defizite in der SoC-Entwicklung, Ursachen für die Defizite und mögliche Lösungen zur Behebung der Ursachen zu bewerten. Dazu wurden die Teilnehmer/-innen für die Fragenkomplexe 1–3 jeweils um eine Priorisierung der fünf Antwortmöglichkeiten in fünf Priorisierungsstufen gebeten.

Die Definition der Priorisierungsstufen erfolgte dabei nach dem Vorbild einer Ratingskala. Dabei wird zwischen einer bipolaren Ratingskala, welche zwei gegensätzliche Dimensionen wie beispielsweise *sehr schlecht* bis *sehr gut* abbildet, und einer unipolaren Ratingskala, auf der lediglich eine Dimension abgebildet wird, unterschieden. Aufgrund der Form der in dieser Arbeit definierten Fragestellungen und Antwortmöglichkeiten für die Fragenkomplexe 1–3 wurde eine Definition der Priorisierungsstufen vergleichbar einer unipolaren Ratingskala gewählt. Das hier gewählte Vorgehen weicht jedoch maßgeblich von einer üblichen Befragung mittels Ratingskala ab. So sollen die Interview-Teilnehmer/-innen nicht, wie es in einer Befragung mittels Ratingskala üblich ist, für jede Fragestellung einen Wert der Ratingskala wählen, sondern die Antwortmöglichkeiten mittels Priorisierung in einer Art Ranking (Rangliste) darstellen. Dabei sollten die Priorisierungsstufen zum einen das Maß der Zustimmung seitens der Interview-Teilnehmer/-innen zur jeweiligen Antwortmöglichkeit im Ranking abbilden, zum anderen wird die Zustimmung, wie nachfolgend gezeigt, in Form einer prozentualen Gewichtung repräsentiert [94].

Priorisierungsstufe 1: (prozentuale Gewichtung: 100 %)
Priorisierungsstufe 1 bedeutet eine Gewichtung der Zustimmung mit 100 % für die Mittelwertberechnung und kann somit als **höchste** bzw. **volle Zustimmung** interpretiert werden.

Priorisierungsstufe 2: (prozentuale Gewichtung: 75 %)
Priorisierungsstufe 2 wird mit einer prozentualen Gewichtung von 75 % in der Mittelwertbildung verrechnet und ist als **hohe Zustimmung** zu interpretieren.
Priorisierungsstufe 3: (prozentuale Gewichtung: 50 %)
Priorisierungsstufe 3 lässt sich als **mittlere Zustimmung** interpretieren und erhält daher die prozentuale Gewichtung von 50 % für die Mittelwertberechnung.
Priorisierungsstufe 4: (prozentuale Gewichtung: 25 %)
Wählt einer/eine der Interview-Teilnehmer/-innen für eine Antwortmöglichkeit die Priorisierungsstufe 4, entspricht dies einer prozentualen Gewichtung für die Mittelwertberechnung von 25 % und lässt sich als **geringe Zustimmung** interpretieren.
Priorisierungsstufe 5: (prozentuale Gewichtung: 0 %)
Für den Fall, dass einer/eine der Interview-Teilnehmer/-innen einer Antwortmöglichkeit **keine Zustimmung** gibt, wird diese als Priorisierungsstufe 5 und damit mit 0 % prozentualer Gewichtung verrechnet.

Durch die Zuordnung von Priorisierungsstufen und prozentualer Gewichtung wurde die Quantifizierung der Zustimmung erreicht. Mithilfe der so erhaltenen prozentualen Werte konnte für jede Antwortmöglichkeit der Fragenkomplexe 1–3, wie in Abbildung 6.2 veranschaulicht, ein Mittelwert über alle Interview-Teilnehmer/-innen hinweg gebildet werden. Der so erhaltene Mittelwert, welcher aus den gewichteten Einzelwerten der Interview-Teilnehmer/-innen berechnet wurde, wird nachfolgend als gewichtete Zustimmungsrate (gZR) bezeichnet.

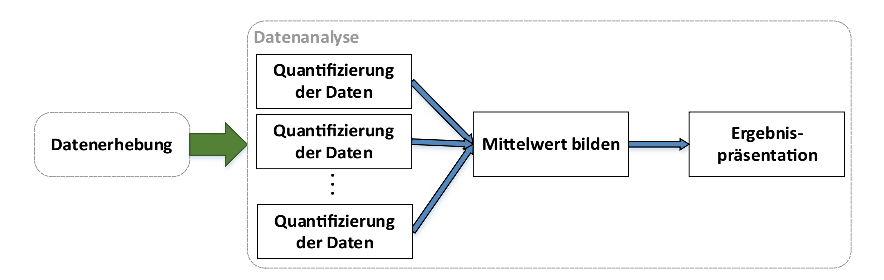


Abbildung 6.2 Datenanalyse Interviewevaluierung

Durch die Berechnung der gewichteten Zustimmungsrate für die jeweilige Antwortmöglichkeit konnte eine repräsentative Priorisierung der Antwortmöglichkeiten aller Teilnehmer/-innen erreicht werden.

Verglichen mit den Fragenkomplexen 1–3 wurde ein separates Vorgehen für die Datenanalyse im Fragenkomplex 4 gewählt. Dieser Unterschied beim Vorgehen der Datenanalyse für den Fragenkomplex 4 resultiert aus einer grundsätzlich anderen Fragestellung. Ziel des Fragenkomplexes 4 ist die Analyse des durch die Interview-Teilnehmer/-innen geschätzten Einflusses der modellbasierten Entwicklungsmethode auf den Aufwand verschiedener Bereiche der SoC-Entwicklung. Daher wurde für den Fragenkomplex 4 eine Befragung mittels bipolarer Ratingskala gewählt. Das bedeutet, den Interview-Teilnehmern/-innen wurden die sieben nachfolgend gezeigten Antwortmöglichkeiten zur Auswahl gestellt. Jede Antwort entspricht dabei einer Aussage über den erwarteten Einfluss der modellbasierten Entwicklungsmethode auf den Aufwand verschiedener Bereiche der SoC-Entwicklung aus Sicht der Interview-Teilnehmer/-innen. Die Interview-Teilnehmer/-innen wurden somit gebeten, eine der Antwortmöglichkeiten auszuwählen. Zusätzlich wurde auch hier eine prozentuale Quantifizierung der Skalenwerte vorgenommen. Die hier definierte Ratingskala teilt sich wie folgt auf:

- stark verringert (> 50 %)
- mittel bis stark verringert (25–50 %)
- leicht verringert (< 25 %)
- unverändert (0 %)
- leicht erhöht (< 25 %)
- mittel bis stark erhöht (25–50 %)
- stark erhöht (> 50 %)

Eine Verringerung bzw. Erhöhung des Aufwandes durch die Anwendung der neuen Methode wurde im Vorfeld auf Basis der leitfadenfreien Interviews auf den Bereich 0 %–50 % geschätzt. Daher wurde die Aufteilung des Wertebereichs mit einer feineren Aufteilung des Bereichs unter 50 % gelegt. Die Zustimmungsrate (ZR) ergibt sich für den Fragenkomplex 4 aus dem prozentualen Anteil der Interview-Teilnehmer/-innen, welche der jeweiligen Antwort und damit der Abschätzung zugestimmt haben. Die Ergebnispräsentation sowie die Diskussion aller Interview-Ergebnisse folgt in Abschnitt 6.1.4.

6.1.3 Interview-Teilnehmer/-innen der leitfadenbasierten Interviews

Im Zuge der in dieser Arbeit durchgeführten Interviewevaluierung wurden zehn Interview-Teilnehmer/-innen der Firma *Robert Bosch GmbH* in Reutlingen aus dem Bereich der SoC-Entwicklung mit verschiedenen Schwerpunkten ausgewählt. Dabei wurde darauf geachtet, dass die Interview-Teilnehmer/-innen bereits mehrere Jahre Erfahrung bei der Anwendung der aktuell bestehenden Entwicklungsmethode aufweisen und zudem bei der Anwendung der neuen modellbasierten Entwicklungsmethode auf ein Pilotprojekt beteiligt waren. Nachfolgend werden in Tabelle 6.1 die zehn Interview-Teilnehmer/-innen auf ihre jeweilige Tätigkeitsfelder aufgeteilt sowie jedem Teilnehmer und jeder Teilnehmerin für die spätere Diskussion der Ergebnisse und Aussagen eine Kennung zugeordnet.

Tabelle 6.1 Interview-Teilnehmer/-innen

Anzahl	Rolle	Kennung
3	Systemarchitekt/-innen	*SA1*, *SA2*, *SA3*
1	Verifikationsingenieur	*VI1*
1	Safety-Manager	*SM1*
4	SoC-Entwickler/-innen	*SE1*, *SE2*, *SE3*, *SE4*
1	Management	*MA1*

Wie bereits beschrieben, wurde bei der Auswahl der Teilnehmer/-innen darauf geachtet, dass die Befragten erste Erfahrungen mit der neuen modellbasierten Entwicklungsmethode erlangt hatten. Darüber hinaus wurde versucht, Entwickler/-innen verschiedener Tätigkeitsfelder auszuwählen, um so unterschiedliche Sichtweisen aus dem jeweiligen Tätigkeitsfeld in Hinblick auf die Evaluierung zu erhalten.

6.1.4 Ergebnisse und Diskussion

Wie bereits zu Beginn des Kapitels beschrieben, wurde vor den eigentlichen Evaluierungsinterviews leitfadenfreie Interviews durchgeführt, um bestehende Defizite in der SoC-Entwicklung sowie deren mögliche Ursachen zu erfragen. Auf Grundlage dieser Interviews wurde anschließend ein Leitfaden für die leifadenbasierte Interviewevaluierung extrahiert. Nachfolgend werden die jeweiligen

Evaluierungsfragen, die Antwortmöglichkeiten sowie die erhaltenen Ergebnisse der Interviewevaluierung vorgestellt. Dabei gibt es vier Abschnitte, welche jeweils einen der vier Fragenkomplexe behandeln. Im ersten Abschnitt werden die Befragung und Analyse der Defizite in der SoC-Entwicklung behandelt. Darauf aufbauend werden im zweiten Abschnitt die Ursachen für das jeweilige Defizit diskutiert. Im nächsten Abschnitt folgen die Beschreibung und die Diskussion der erhaltenen Ergebnisse zu den Lösungsansätzen des jeweiligen Defizits. Im letzten Abschnitt wird der Einfluss der modellbasierten Entwicklungsmethode auf den Aufwand in verschiedenen Bereichen sowie der gesamten SoC-Entwicklung behandelt.

Dadurch sollte die nötige Transparenz für die Beantwortung der Fragen geschaffen und den Interview-Teilnehmern/-innen zudem bei der Abgabe einer Antwort geholfen werden. Besonders bei der Beantwortung der Fragen aus dem Fragenkomplex 4, erleichterte die Zuordnung von prozentualen Werten zu den Antwortmöglichkeiten der Ratingskala die Abschätzung des Einflusses. Denn bei diesen Fragen wurden die Interview-Teilnehmer/-innen aufgefordert, eine Abschätzung über den Einfluss der modellbasierten Entwicklungsmethode auf den Aufwand bezogen auf die Teilbereiche und den gesamten SoC-Entwicklungs-Flow abzugeben.

Ergebnis und Diskussion: Defizite in der SoC-Entwicklung

Frage 1 befasst sich mit der Priorisierung der Defizite in Hinblick auf die negative Auswirkung des jeweiligen Defizites auf die Effizienz der SoC-Entwicklung. Dabei wurden aus dem Leitfaden fünf mögliche Antworten zur Verfügung gestellt, um deren Priorisierung die Interview-Teilnehmer/-innen von „höchste Zustimmung“ und damit „höchster negativer Einfluss“ nach „keine Zustimmung“ bzw. „kein negativer Einfluss“, wie in Abschnitt 6.1.1 und Abschnitt 6.1.2 beschrieben, gebeten wurden. Tabelle 6.2 zeigt die genaue Fragestellung, die zur Verfügung stehenden Antwortmöglichkeiten sowie die als Ergebnis der Befragung und anschließenden Analyse erhaltene gZR der Interviewevaluierung.

D1 (Implementierungsfehler) und *D2* (Verifikationsfehler) fokussieren sich auf die direkten Auswirkungen einer defizitären Spezifikation, auf die Implementierung bzw. die Verifikation und damit auf die Auswirkung des Defizites auf die Effizienz aufgrund der benötigten Fehlerbehebung.

Bei der Interviewevaluierung wurden Implementierungsfehler aufgrund einer unzureichenden Spezifikation mit 23 % gZR und Verifikationsfehler aufgrund einer unzureichenden Spezifikation mit 20 % gZR am niedrigsten priorisiert. Die Befragten *SE1*, *SE2*, *SM1* und *SE3* gaben zusätzlich an, dass aus ihrer Sicht entstandene Fehler in der Implementierung durch die Verifikation im aktuell bestehenden

Tabelle 6.2 Fragestellung, Antwortmöglichkeiten und Ergebnis Frage 1

Fragestellung	Antwortmöglichkeiten	gZR	Kennung
Welches der hier gezeigten Defizite hat den stärksten negativen Einfluss auf die Effizienz der SoC-Entwicklung? Bitte priorisieren Sie nach Stärke des Einflusses.	Implementierungsfehler aufgrund unzureichender Spezifikation	23 %	*D1*
	Verifikationsfehler aufgrund unzureichender Spezifikation	20 %	*D2*
	Nachträgliche Änderungen an Lasten- bzw. Pflichtenheft	80 %	*D3*
	Inkonsistenzen zwischen Spezifikationsdokumenten sowie zwischen Spezifikation und Entwurf	65 %	*D4*
	Fehlkommunikation zwischen Kunde und Entwickler sowie zwischen den Entwicklern/-innen	63 %	*D5*

Entwicklungs-Flow erkannt und behoben werden. Zudem wird bereits von Anfang an eine Behebung der Fehler in der Planung berücksichtigt.

D3 steht für den negativen Einfluss auf die Effizienz durch nachträgliche Änderungen an den vereinbarten Anforderungen zwischen Kunde und Entwickler. Solche nachträglichen Änderungen werden meist erst spät erkannt und deren Durchführung bindet in der Regel eine hohe Menge Ressourcen, führt zu einer Verschiebung des Zeitplans und kann laut *SE1* und *SE2* zu Folgefehlern führen. Die Folgefehler entstehen dabei aufgrund von nicht beachteten Einflüssen auf das umgebende System, welche durch die nachträglichen Änderungen hervorgerufen werden. Aufgrund der Häufigkeit, mit der das Defizit *D3* in der SoC-Entwicklung auftritt, und die schwerwiegenden Auswirkungen auf die Effizienz wurde das Defizit *D3* mit 80 % gZR am höchsten priorisiert.

Dem folgen *D4* (Inkonsistenzen Spezifikation/Entwurf) mit 65 % gZR und *D5* (Fehlkommunikation Entwickler-Kunde) mit 63 % gZR. Dabei wurde *D4* von den Interview-Teilnehmern/-innen *SA2*, *SM1* und *SE3* am höchsten priorisiert. Die stellenweise hohe Priorisierung resultiert laut *SE3* aus dem häufigen Auftreten des Defizits und der Gefahr, dass durch die Inkonsistenz Fehler im Entwurf durch die Verifikation nicht erkannt werden könnten.

D5 erhielt von fast allen Interview-Teilnehmern/-innen eine Priorisierung zwischen Priorisierungsstufe 1 und Priorisierungsstufe 3. Zudem wurde es besonders von den Systemarchitekten/-innen und dem Interview-Teilnehmer aus dem Managementbereich hoch priorisiert. Als Grund wurden dabei von *MA1* auch hier die

schwerwiegenden Folgen aus solchen Fehlkommunikationen für ein Entwicklungsprojekt genannt.

Aus den Interviews ging hervor, dass alle der hier behandelten Defizite aktuell in der SoC-Entwicklung bestehen und die Effizienz der Entwicklung beeinträchtigen. Da jedoch die Defizite *D3–D5* aus Sicht und Erfahrung der Befragten zu besonders umfangreichen Nacharbeiten führen können, ist deren Einfluss höher zu priorisieren.

Ergebnis und Diskussion: Ursachen

Im nächsten Schritt der Interviewevaluierung wurden die Teilnehmer/-innen zu den Ursachen für das jeweilige Defizit befragt. Dabei wurden ebenfalls Antwortmöglichkeiten als Teil des Leitfadens für die Priorisierung zur Verfügung gestellt, welche als mögliche Ursachen auf Basis der leitfadenfreien Interviews identifiziert wurden. Ziel dieser Befragung war es, festzustellen, ob der Ursprung der Defizite *D1–D5* auf eine defizitäre Entwicklungs- bzw. Spezifikationsmethodik zurückzuführen ist. Dabei soll der Einfluss der als Antwortmöglichkeiten bereitgestellten Ursachen auf die Entstehung des Defizits priorisiert werden. Die Priorisierung als „höchste Zustimmung“ (Priorisierungsstufe 1) entspricht somit der Einstufung als Hauptursache. Die Priorisierung als Priorisierungsstufe 5 („keine Zustimmung“) entspricht der Einschätzung, dass die Ursache nicht oder im Vergleich zu den übrigen Ursachen einen vernachlässigbaren Einfluss auf die Entstehung des Defizits hat.

Tabelle 6.3 zeigt die Fragestellung, die Antwortmöglichkeiten sowie die gZR als Ergebnis der Befragung bezüglich möglicher Ursachen für *D1*.

Tabelle 6.3 Fragestellung, Antwortmöglichkeiten und Ergebnis Frage 2.1

Fragestellung	Antwortmöglichkeiten	gZR	Kennung
Die hier gezeigten Ursachen *U1–U5* führen zum Defizit *D1* (Implementierungsfehler aufgrund unzureichender Spezifikation). Inwieweit stimmen Sie den Antwortmöglichkeiten zu? Bitte priorisieren Sie.	Unvollständige Spezifikation	*75 %*	*U1*
	Fehlende Eindeutigkeit der Spezifikation	*70 %*	*U2*
	Mangelnde Verständlichkeit/Lesbarkeit der Spezifikation	*15 %*	*U3*
	Fehlende Nachverfolgbarkeit der Anforderungen	*35 %*	*U4*
	Manueller Übertrag innerhalb der Spezifikationsdokumente sowie zwischen Spezifikation und Entwurf	*55 %*	*U5*

Die Interview-Teilnehmer/-innen wurden gebeten, die in der Tabelle 6.3 dargestellten Antwortmöglichkeiten zu nennen, welche aus ihrer Sicht als Ursache für das Defizit *D1* besteht. Da in der Regel mehrere Ursachen für die hier behandelten Defizite verantwortlich sind, wurden die Interview-Teilnehmer/-innen zudem gebeten, die Ursachen zu priorisieren. Nach der Befragung wurden die Antworten gemäß Abschnitt 6.1.2 gewichtet und die gZR berechnet.

Wie in der Tabelle 6.3 zu sehen, wurde die Unvollständigkeit der Spezifikation (*U1*) als Hauptursache für Implementierungsfehler mit einer gZR von 75 % priorisiert. So führt eine Unvollständigkeit der Spezifikation laut *SA2* und *SE1* dazu, dass die Modulentwickler/-innen die fehlenden Informationen durch Annahmen ergänzen und es in Folge dessen häufig zu Implementierungsfehlern kommen kann.

Die zweithöchste Priorisierung mit 70 % gZR bei der Interviewevaluierung erhielt die fehlende Eindeutigkeit der Spezifikation (*U2*) als Ursache für Implementierungsfehler. Ähnlich wie bei der Unvollständigkeit der Spezifikation sind auch hier die Entwickler/-innen gezwungen, die Informationen zu interpretieren und somit Annahmen zu treffen, welche zu Fehlern bei der Implementierung führen können.

Die *U3* (Mangelnde Verständlichkeit) mit 15 % gZR und *U4* (fehlende Nachverfolgbarkeit) mit 35 % gZR wurden von allen Interview-Teilnehmern/-innen gering priorisiert. Als Grund hierfür nannten die Interview-Teilnehmer/-innen *SE2* und *SE3*, dass sowohl die Verständlichkeit aus Sicht der Modulentwickler/-innen als auch die Nachverfolgbarkeit der Anforderungen bereits im aktuell angewendeten Entwicklungs-Flow gegeben sei.

Der manuelle Übertrag (*U5*) erhielt dabei von drei Interview-Teilnehmern/-innen die niedrigste und von vier Interview-Teilnehmern/-innen die höchste Priorisierung und kam damit auf ein gZR von 55 %. Hier lassen sich zwei Sichtweisen aus den Interviews extrahieren. Der manuelle Übertrag der Informationen zwischen Spezifikation und Entwurf führt häufig zu Fehlern bei der Implementierung und wurde daher vereinzelt hoch priorisiert. Dementgegen werden die Fehler bei der Verifikation entdeckt und verursachen somit meist nur geringe Auswirkungen auf die Effizienz der Entwicklung.

Das gleiche Vorgehen wurde für die Frage nach den Ursachen für *D2* gewählt. Den Interview-Teilnehmern/-innen wurden dieselben Antwortmöglichkeiten, wie in Tabelle 6.4 zu sehen, zur Verfügung gestellt. Erneut zeigt die Tabelle 6.4 die gZR als Ergebnis der Befragung.

Als Hauptursache mit 80 % gZR nannten die Interview-Teilnehmer/-innen eine unvollständige Spezifikation (*U1*). Eine unvollständige Spezifikation führt – wie bei der Implementierung – bei der Verifikation dazu, dass die Verifikationsingenieure/-innen zu Annahmen gezwungen sind, um die fehlenden Informationen zu ergänzen.

Tabelle 6.4 Fragestellung, Antwortmöglichkeiten und Ergebnis Frage 2.2

Fragestellung	Antwortmöglichkeiten	gZR	Kennung
Die hier gezeigten Ursachen *U1–U5* führen zum Defizit *D2* (Verifikationsfehler aufgrund unzureichender Spezifikation). Inwieweit stimmen Sie den Antwortmöglichkeiten zu? Bitte priorisieren Sie.	Unvollständige Spezifikation	*80 %*	*U1*
	Fehlende Eindeutigkeit der Spezifikation	*65 %*	*U2*
	Mangelnde Verständlichkeit/Lesbarkeit der Spezifikation	*38 %*	*U3*
	Fehlende Nachverfolgbarkeit der Anforderungen	*38 %*	*U4*
	Manueller Übertrag innerhalb der Spezifikationsdokumente sowie zwischen Spezifikation und Verifikation	*30 %*	*U5*

Dieser Sachverhalt birgt bei der Verifikation zudem die Gefahr, dass Fehler in der Implementierung durch eine defizitäre Verifikation nicht oder sehr spät erkannt und erhebliche Aufwände für die Fehlerbehebung verursacht werden. Ähnlich verhält es sich bei einer nicht eindeutigen Spezifikation (*U2*). Diese Ursache erhielt eine gZR von 65 % und birgt ebenfalls durch die Gefahr von Fehlinterpretationen das Risiko, dass die Implementierung nicht korrekt verifiziert wird und somit Fehler nicht erkannt werden können.

Die mangelnde Verständlichkeit (*U3*) und die fehlende Nachverfolgbarkeit der Anforderungen (*U4*) wurden mit einer gZR von 38 % priorisiert. Auch hier wurden die bestehende Verständlichkeit sowie die bestehende Nachverfolgbarkeit der Anforderungen als Begründung für die Priorisierung genannt. Darüber hinaus wurde die *U4* dreimal als höchst priorisierte und viermal als niedrig priorisierte Ursache genannt. In der weiteren Befragung zu den Gründen der Priorisierung wurden die unterschiedlichen Arbeitsweisen und Erfahrungen der Interview-Teilnehmer/-innen deutlich. Zum einen wurde die Ursache *U4* hoch priorisiert, da eine fehlende Nachverfolgbarkeit der Anforderungen aktuell zu erheblichen Problemen bei der Verifikation führt. So betonten alle Interview-Teilnehmer/-innen die Wichtigkeit der Nachverfolgbarkeit der Anforderungen. Darüber hinaus besteht laut *SE4* aktuell keine Zeit, um diese in den Projekten umzusetzen. Demgegenüber gaben die Befragten *SE2* und *SE3* an, dass die Nachverfolgbarkeit der Anforderungen in aktuellen Projekten bereits in hohem Maße gegeben sei und priorisierten daher *U4* niedrig ein.

Die geringste gZR mit 30 % erhielt der manuelle Übertrag als Ursache für Fehler bei der Verifikation. Zudem wurde hier seitens des *VI1* die aus seiner Sicht hohe Wichtigkeit einer manuellen Erstellung der Verifikation genannt, um diese erfolgreich durchführen zu können und die Übertragung von Fehlern aus der Spezifikation in die Verifikation zu verhindern.

Vergleichbar fiel die Priorisierung der Ursachen für nachträgliche Änderungen (*D3*) aus, wie Tabelle 6.5 zeigt.

Tabelle 6.5 Fragestellung, Antwortmöglichkeiten und Ergebnis Frage 2.3

Fragestellung	Antwortmöglichkeiten	gZR	Kennung
Die hier gezeigten Ursachen *U1–U5* führen zum Defizit *D3* (Nachträgliche Änderungen an Lasten- bzw. Pflichtenheft). Inwieweit stimmen Sie den Antwortmöglichkeiten zu? Bitte priorisieren Sie.	Unvollständige Spezifikation	*83 %*	*U1*
	Fehlende Eindeutigkeit der Spezifikation	*75 %*	*U2*
	Mangelnde Verständlichkeit/Lesbarkeit der Spezifikation	*53 %*	*U3*
	Fehlende Nachverfolgbarkeit der Anforderungen	*40 %*	*U4*
	Manueller Übertrag innerhalb der Spezifikationsdokumente sowie zwischen Spezifikation und Entwurf	*8 %*	*U5*

Auch hier wurde die Unvollständigkeit der Spezifikation (*U1*) mit 83 % gZR als Hauptursache für nachträgliche Änderungen genannt, da besonders eine fehlende Vollständigkeit in Pflichtenheft und Systemkonzept laut *SE1*, *SE3* und *SA1* zu einer fehlenden Realisierung von Funktionalitäten im SoC führen kann.

Vergleichbar verhält es sich bei der fehlenden Eindeutigkeit der Spezifikation (*U2*), da dies zu Fehlinterpretationen und damit gegebenenfalls sogar zu einer Realisierung falscher Funktionen führen könnte, welche so nicht vom Kunden gefordert wurden. *U2* wurde mit einer gZR von 75 % priorisiert.

Die mangelnde Verständlichkeit der Spezifikation (*U3*) wurde mit 53 % als Ursache für nachträgliche Änderungen priorisiert. Dabei birgt die mangelnde Verständlichkeit laut *SM1* besonders die Gefahr, dass der Kunde das Pflichtenheft bei der Planungsphase nicht korrekt versteht und somit ein fehlerbehaftetes Verständnis des Entwicklungsvorhabens erhält.

Die fehlende Nachverfolgbarkeit der Anforderungen (*U4*) wurde in diesem Zusammenhang oft als wichtig herausgestellt, da besonders die Nachverfolgbarkeit

zwischen Kundenanforderungen und Systemanforderungen helfen kann, nachträgliche Änderungen zu vermeiden. Die geringe Priorisierung von 40 % resultiert hierbei erneut aus der unterschiedlichen Einschätzung der Interview-Teilnehmer/-innen über die aktuell bestehende Nachverfolgbarkeit der Anforderungen in der SoC-Entwicklung. Eine sehr eindeutig niedrige Priorisierung mit 8 % gZR erhielt der manuelle Übertrag als Ursache für nachträgliche Änderungen.

Das Ergebnis der Befragung über die Ursachen für Inkonsistenzen Spezifikation/Entwurf sowie die Fragestellung und Antwortmöglichkeiten sind in Tabelle 6.6 dargestellt.

Tabelle 6.6 Fragestellung, Antwortmöglichkeiten und Ergebnis Frage 2.4

Fragestellung	Antwortmöglichkeiten	gZR	Kennung
Die hier gezeigten Ursachen *U1–U5* führen zum Defizit *D4* (Inkonsistenzen Spezifikation/Entwurf). Inwieweit stimmen Sie den Antwortmöglichkeiten zu? Bitte priorisieren Sie.	Unvollständige Spezifikation	*70 %*	*U1*
	Fehlende Eindeutigkeit der Spezifikation	*78 %*	*U2*
	Mangelnde Verständlichkeit/Lesbarkeit der Spezifikation	*18 %*	*U3*
	Fehlende Nachverfolgbarkeit der Anforderungen	*35 %*	*U4*
	Manueller Übertrag innerhalb der Spezifikationsdokumente sowie zwischen Spezifikation und Entwurf	*50 %*	*U5*

Bei der Frage nach den Ursachen für Inkonsistenzen zwischen den Spezifikationsdokumenten sowie zwischen Spezifikation und Entwurf (*D4*) wurde die fehlende Vollständigkeit (*U1*) als zweithöchste priorisierte Ursache mit 70 % gZR genannt. Die fehlende Eindeutigkeit der Spezifikation (*U2*) ist laut den Befragten mit 78 % gZR Hauptursache für Inkonsistenzen. Dabei wurde vermehrt von Situationen berichtet, bei denen in der Entwurfsphase der SoC auf Grundlage des Entwurfes weiterentwickelt wurde und nicht auf Grundlage der Spezifikation. Darüber hinaus wurden die so getroffenen Entwurfsentscheidungen nicht oder nur unzureichend in der Spezifikation nachgepflegt.

Die mangelnde Verständlichkeit der Spezifikation (*U3*) wurde mit einer gZR von 18 % niedrig priorisiert. Die mangelnde Verständlichkeit ist aus Sicht der Interview-Teilnehmer/-innen und damit aus Sicht der Entwickler/-innen eher selten der Fall im aktuell angewendeten Entwicklungs-Flow. Die fehlende Nachverfolgbarkeit der

Anforderungen (*U4*) wurde mit 35 % gZR priorisiert. Der manuelle Übertrag erhielt somit die dritthöchste Priorisierung mit 50 % gZR.

Bei der Befragung zu den möglichen Ursachen für die Fehlkommunikation zwischen Kunde und Entwickler wurden den Interview-Teilnehmern/-innen lediglich die Antwortmöglichkeiten Ursache *U1–U4* zur Verfügung gestellt, da die *U5* (manueller Übertrag) nicht als mögliche Ursache für die Fehlkommunikation mit dem Kunden infrage kommt (Tabelle 6.7).

Tabelle 6.7 Fragestellung, Antwortmöglichkeiten und Ergebnis Frage 2.5

Fragestellung	Antwortmöglichkeiten	gZR	Kennung
Die hier gezeigten Ursachen *U1–U5* führen zum Defizit *D5* (Fehlkommunikation Kunde-Entwickler). Inwieweit stimmen Sie den Antwortmöglichkeiten zu? Bitte priorisieren Sie.	Unvollständige Spezifikation	*65 %*	*U1*
	Fehlende Eindeutigkeit der Spezifikation	*80 %*	*U2*
	Mangelnde Verständlichkeit/Lesbarkeit der Spezifikation	*60 %*	*U3*
	Fehlende Nachverfolgbarkeit der Anforderungen	*35 %*	*U4*

Als Ursache für die Fehlkommunikation zwischen Kunde und Entwickler erhielt die unvollständige Spezifikation eine gZR von 65 % und ist damit die zweithöchst priorisierte Ursache für die Fehlkommunikation. Als Hauptursache mit 80 % gZR lässt sich die fehlende Eindeutigkeit der Spezifikation als Ergebnis der Befragung identifizieren. Auch die mangelnde Verständlichkeit erhielt in vielen Fällen eine hohe Priorisierung und kommt insgesamt auf 60 % gZR. Somit liegen die Ursachen *U1–U4* bei diesem Ergebnis besonders nah beieinander. Die fehlende Nachverfolgbarkeit erhielt als niedrig priorisierte Ursache 35 % gZR.

Zusammenfassend lassen sich als Hauptursachen für die hier betrachteten Defizite besonders die Unvollständigkeit und die fehlende Eindeutigkeit der Spezifikation herausstellen, da diese durchweg eine vergleichsweise hohe Priorisierung erhielten.

Ergebnis und Diskussion Lösungsansätze

Nach der Analyse zu möglichen Ursachen für die Defizite in der SoC-Entwicklung folgte die Befragung der Interview-Teilnehmer/-innen zu möglichen Lösungsansätzen, um die Defizite *D1* bis *D5* zu minimieren bzw. die Ursachen für die Defizite anzugehen. Wie bei den Fragen zuvor, standen auch hier den Interview-Teilnehmern/-innen unterschiedliche Antwortmöglichkeiten als Teil des Leitfadens

zur Verfügung. Dabei wurden die Interview-Teilnehmer/-innen erneut gebeten, die Lösungsansätze zu priorisieren. Hierbei bestanden einzelne der Antwortmöglichkeiten aus Teilaspekten der in dieser Arbeit entwickelten Gesamtmethode. So sind beispielsweise die Lösungsansätze „Modellbasierte Spezifikation" und „Formalisierung Spezifikation" Teil der hier vorliegenden Arbeit und wurden in Abschnitt 4.1 und Abschnitt 5.1 näher behandelt. Ebenso beinhaltet die Arbeit den „modellgetriebenen Entwurf", eine „starke Nachverfolgbarkeit der Anforderungen" und die „Arbeitsweise nach dem Top-down-Ansatz". Die „Ausführbarkeit der Spezifikation" ist jedoch in dieser Arbeit nur teilweise erfüllt und könnte daher in einer weiterführenden Arbeit Umsetzung finden.

Die Antworten wurden nach der Interviewevaluierung, wie zuvor in Abschnitt 6.1.2 beschrieben, quantifiziert und ein Mittelwert über alle Antworten der Interview-Teilnehmer/-innen gebildet. Tabelle 6.8 beinhaltet die Fragestellung sowie Antwortmöglichkeiten und das Ergebnis für die Frage 3.1. Die Fragestellung bleibt hierbei für alle Defizite gleich, die Antwortmöglichkeiten variieren jedoch.

Tabelle 6.8 Fragestellung, Antwortmöglichkeiten und Ergebnis Frage 3.1

Fragestellung	**Antwortmöglichkeiten**	**gZR**
Die hier angebotenen Lösungsansätze reduzieren bzw. beheben das Defizit D1 (Implementierungsfehler aufgrund unzureichender Spezifikation). Inwieweit stimmen Sie den Antwortmöglichkeiten zu? Bitte priorisieren Sie.	Modellbasierte Spezifikation	*78 %*
	Formalisierten Spezifikation	*75 %*
	Ausführbare Spezifikation	*28 %*
	Modellgetriebener Entwurf	*63 %*
	Stärkere Nachverfolgbarkeit der Anforderungen	*8 %*

Die höchste Priorisierung als möglichen Lösungsansatz für „Implementierungsfehler aufgrund einer unzureichenden Spezifikation" wurde die modellbasierte Spezifikation mit 78 % gZR, dicht gefolgt von der formalisierten Spezifikation mit 75 % gZR genannt. Eine hohe gZR mit 63 % erhielt zudem der modellgetriebene Entwurf als Lösungsansatz. Die ausführbare Spezifikation mit 28 % und besonders die stärkere Nachverfolgbarkeit wurden bezogen auf das Vorbeugen von Implementierungsfehlern niedrig priorisiert.

Vergleicht man dieses Ergebnis mit den Ergebnissen aus der Befragung zu den Ursachen des Defizits *D1*, lässt sich eine Zuordnung ableiten. Als Hauptursachen wurden die Unvollständigkeit sowie die fehlende Eindeutigkeit der Spezifikation angegeben, als Lösungen für dasselbe Defizit die modellbasierte Spezifikation und

die formalisierte Spezifikation. Diese Lösungsansätze haben das Potenzial, die Vollständigkeit und Eindeutigkeit der Spezifikation signifikant zu erhöhen. Zu dieser Einschätzung kamen somit auch die Interview-Teilnehmer/-innen. Als dritte Ursache wurde der manuelle Übertrag priorisiert, und so liegt es nahe, dass auch die Automatisierung der Implementierung mittels modellgetriebenem Entwurf bei der Frage nach möglichen Lösungsansätzen hoch priorisiert wird.

Bei Frage 3.2 zu möglichen Lösungsansätzen und deren Wirkung für das Defizit *D2* (Verifikationsfehler aufgrund unzureichender Spezifikation) blieben die Antwortmöglichkeiten weitestgehend gleich. Lediglich der modellgetriebene Entwurf wurde durch die modellgetriebene Verifikation, welche ebenfalls Teil dieser Arbeit ist, ersetzt. Tabelle 6.9 zeigt die Fragestellung sowie die Antwortmöglichkeiten und das Ergebnis der Befragung zu Frage 3.2.

Tabelle 6.9 Fragestellung, Antwortmöglichkeiten und Ergebnis Frage 3.2

Fragestellung	Antwortmöglichkeiten	gZR
Die hier angebotenen Lösungsansätze reduzieren bzw. beheben das Defizit *D2* (Verifikationsfehler aufgrund unzureichender Spezifikation). Inwieweit stimmen Sie den Antwortmöglichkeiten zu? Bitte priorisieren Sie.	Modellbasierte Spezifikation	*53 %*
	Formalisierten Spezifikation	*75 %*
	Ausführbare Spezifikation	*39 %*
	Modellgetriebene Verifikation	*44 %*
	Stärkere Nachverfolgbarkeit der Anforderungen	*39 %*

Als möglicher Lösungsansatz für die Reduzierung von Verifikationsfehlern aufgrund einer unzureichenden Spezifikation wurde die Formalisierung der Spezifikation von den Interview-Teilnehmern/-innen mit 75 % gZR höchst priorisiert. Die Modellierung der Spezifikation mit 53 % gZR wurde als zweithöchste Priorisierung benannt. Die ausführbare Spezifikation (39 % gZR), die modellgetriebene Verifikation (44 % gZR) und die stärkere Nachverfolgbarkeit (39 % gZR) lagen nah beieinander und erhielten mit etwa 40 % ebenfalls eine verhältnismäßig hohe gZR.

Für die Befragung nach möglichen Lösungsansätzen, um nachträglichen Änderungen vorzubeugen, wurden die Antwortmöglichkeiten erneut entsprechend angepasst, wie in Tabelle 6.10 zu sehen. Die Tabelle 6.10 zeigt zudem die Fragestellung sowie das Ergebnis der Befragung in Form der gZR.

Bei der Frage, welche der hier gezeigten Lösungsansätze das Auftreten von nachträglichen Änderungen in der SoC-Entwicklung zu reduzieren vermag, wurde mit hoher Übereinstimmung der Top-down-Ansatz mit 85 % gZR höchst priorisiert. Durch die ausführliche Analyse der Kundenanforderungen und Anwendungsfälle

Tabelle 6.10 Fragestellung, Antwortmöglichkeiten und Ergebnis Frage 3.3

Fragestellung	Antwortmöglichkeiten	gZR
Die hier angebotenen Lösungsansätze reduzieren bzw. beheben das Defizit *D3* (Nachträgliche Änderungen an Lasten- bzw. Pflichtenheft). Inwieweit stimmen Sie den Antwortmöglichkeiten zu? Bitte priorisieren Sie.	Modellbasierte Spezifikation	*65 %*
	Formalisierte Spezifikation	*60 %*
	Ausführbare Spezifikation	*25 %*
	Top-down-Ansatz	*85 %*
	Stärkere Nachverfolgbarkeit der Anforderungen	*20 %*

sowie die lösungslosgelöste Dekomposition der benötigten Funktionen sinkt die Wahrscheinlichkeit, geforderte Funktionen zu vernachlässigen bzw. nicht geforderte Funktionen dennoch zu realisieren. Darüber hinaus steigt die Vollständigkeit der Spezifikation durch den Top-down-Ansatz in der Regel an.

Die modellbasierte Spezifikation mit 65 % gZR und die formalisierte Spezifikation mit 60 % gZR wurden ebenfalls als mögliche Lösungen hoch priorisiert. So sind sich die Interview-Teilnehmer/-innen einig, dass die Modellierung und Formalisierung der Spezifikation zu einer Steigerung der Spezifikationsqualität führt und somit nachträgliche Änderungen verhindert werden können. Die ausführbare Spezifikation (25 % gZR) und die stärkere Nachverfolgbarkeit der Anforderungen (20 % gZR) wurden verhältnismäßig niedrig priorisiert.

Das Ergebnis der Befragung zu möglichen Lösungsansätzen für die Reduzierung von Inkonsistenzen Spezifikation/Entwurf, die dabei verwendete Fragestellung sowie die Antwortmöglichkeiten sind nachfolgend in Tabelle 6.11 abgebildet.

Tabelle 6.11 Fragestellung, Antwortmöglichkeiten und Ergebnis Frage 3.4

Fragestellung	Antwortmöglichkeiten	gZR
Die hier angebotenen Lösungsansätze reduzieren bzw. beheben das Defizit *D4* (Inkonsistenzen Spezifikation/Entwurf). Inwieweit stimmen Sie den Antwortmöglichkeiten zu? Bitte priorisieren Sie.	Modellbasierte Spezifikation	*65 %*
	Formalisierte Spezifikation	*70 %*
	Ausführbare Spezifikation	*8 %*
	Modellgetriebener Entwurf	*55 %*
	Top-down-Ansatz	*53 %*

Um Inkonsistenzen zwischen Spezifikation und Entwurf vorzubeugen, wurde durch die Interview-Teilnehmer/-innen ebenfalls die Formalisierung der Spezifikation als höchst priorisierter Lösungsansatz mit 70 % gZR benannt. Dicht gefolgt

jedoch von der Modellierung der Spezifikation mit 65 % gZR, dem modellgetriebenen Entwurf mit 55 % gZR und dem Top-down-Ansatz mit 53 % gZR. Die gleichmäßige Verteilung könnte darauf hinweisen, dass ein kombinierter Ansatz, wie er in dieser Arbeit vorliegt, benötigt wird, um das Auftreten von Inkonsistenzen zu reduzieren. Als Ursache für das Defizit *D4* wurde die fehlende Eindeutigkeit, gefolgt von der Unvollständigkeit und dem manuellen Übertrag genannt. Diese Ursachen lassen sich am stärksten durch die Modellierung und Formalisierung der Spezifikation sowie durch die Automatisierung mittels modellgetriebenem Entwurf reduzieren. Die Ausführbarkeit der Spezifikation wurde in dieser Befragung sehr gering mit nur 8 % gZR priorisiert.

Tabelle 6.12 zeigt zum Schluss des Abschnittes die Fragestellung, mögliche Lösungsansätze sowie das Ergebnis der Befragung in Form der gZR. Befragt wurden die Interview-Teilnehmer/-innen zu möglichen Lösungsansätzen für die Reduzierung von Fehlkommunikationen zwischen Kunde und Entwickler sowie zwischen den Entwicklern/-innen.

Tabelle 6.12 Fragestellung, Antwortmöglichkeiten und Ergebnis Frage 3.5

Fragestellung	**Antwortmöglichkeiten**	**gZR**
Die hier angebotenen Lösungsansätze reduzieren bzw. beheben das Defizit *D5* (Fehlkommunikation Kunde-Entwickler). Inwieweit stimmen Sie den Antwortmöglichkeiten zu? Bitte priorisieren Sie.	Modellbasierte Spezifikation	*65 %*
	Formalisierte Spezifikation	*70 %*
	Ausführbare Spezifikation	*68 %*
	Stärkere Nachverfolgbarkeit Anforderungen	*48 %*

Wie zuvor kam auch bei dieser Frage eine nur knapp unterschiedliche Priorisierung zustande. Höchst priorisiert wurde die Formalisierung der Spezifikation mit 70 % gZR. Mit nur 2 % Unterschied folgte in diesem Fall die ausführbare Spezifikation mit 68 % gZR. Die modellbasierte Spezifikation erhielt eine gZR von 65 % und die stärkere Nachverfolgbarkeit der Anforderungen eine gZR von 48 %.

Dies ist die höchste Priorisierung der ausführbaren Spezifikation über das gesamte Evaluierungssinterview betrachtet. So sehen die Interview-Teilnehmer/-innen besonders bei der Kommunikation mit dem Kunden große Vorteile in der Ausführbarkeit der Spezifikation, um so die Verständlichkeit der Spezifikation neben der Formalisierung und Modellierung weiter zu erhöhen.

Bei der, im vorangegangenen Abschnitt beschriebenen Evaluierung möglicher Ursachen der nachfolgend näher betrachteten Defizite erhielten besonders die

Lösungsansätze Modellierung und Formalisierung der Spezifikation sowie Top-down-Ansatz eine durchweg hohe Priorisierung. Somit konnte der Nutzen der in dieser Arbeit vorgestellten modellbasierten Entwicklungsmethode auf die Reduzierung der Defizite und damit die Steigerung der Effizienz in der SoC-Entwicklung aus Sicht der Interview-Teilnehmer/-innen evaluiert und bestätigt werden.

Für die Kommunikation mit dem Kunden und um dabei Missverständnisse auszuschließen, sollte jedoch die Ausführbarkeit großer Teile der Spezifikation zusätzlich ermöglicht werden.

Bewertung der Aufwände

Zum Ende der Interviewevaluierung wurden die Teilnehmer/-innen gebeten, den Einfluss der neuen modellbasierten Methode auf die Aufwände in den nachfolgenden Bereichen im Vergleich zur aktuellen Arbeitsweise zu bewerten:

- Spezifikation
- Verifikation
- Entwurf
- SoC-Entwicklung gesamt

Zusätzlich wird die prozentuale Verringerung bzw. Erhöhung auf den durchschnittlichen Aufwand eines SoCs bezogen. Dazu wird ein gemittelter Wert von 20 Mio. Euro Entwicklungskosten für einen kundenspezifischen Automotive SoC als Grundlage genommen. Dieser Wert resultiert aus aktuellen Werten der SoC-Entwicklung der Firma *Robert Bosch GmbH*. Darüber hinaus teilen sich für die hier vorliegende Arbeit die Kosten wie folgt auf: Für die Spezifikation sind bei dem hier zugrunde liegenden Beispiel 3 Mio. Euro, für den Entwurf 5 Mio. Euro und für die Verifikation weitere 5 Mio. Euro vorgesehen. Die verbleibenden 7 Mio. Euro werden für Posten wie Projektplanung, Fehlerbehebung und Maskendruck benötigt. Aufgrund der großen Varianz bei der Komplexität und beim Umfang der SoCs handelt es sich hierbei lediglich um an die Realität angenäherte Werte. Neben den finanziellen Kosten wird eine durchschnittliche Entwicklungszeit von vier Jahren als Basis gewählt.

Tabelle 6.13 zeigt das Ergebnis der Befragung über den Einfluss der modellbasierten Entwicklung auf den Aufwand bei der Spezifikation. Dabei gaben alle Interview-Teilnehmer/-innen im ersten Schritt als Tendenz eine Erhöhung des Aufwandes an.

Die Tabelle 6.13 zeigt die Einstimmigkeit bezüglich der Tendenz aller Interview-Teilnehmer/-innen, dass sich der Aufwand im Bereich der Spezifikation durch Anwendung der neuen Methode auf die SoC-Entwicklung erhöhen wird. Dabei

Tabelle 6.13 Fragestellung, Antwortmöglichkeiten und Ergebnis Frage 4.1

Fragestellung	Antwortmöglichkeit	ZR
Wie wirkt sich die modellbasierte Entwicklungsmethode auf den Aufwand bei der Spezifikation, verglichen mit der aktuell angewendeten Arbeitsweise aus?	stark verringert (> 50 %)	*0 %*
	mittel bis stark verringert (25–50 %)	*0 %*
	leicht verringert (< 25 %)	*0 %*
	unverändert	*0 %*
	leicht erhöht (< 25 %)	*20 %*
	mittel bis stark erhöht (25–50 %)	*60 %*
	stark erhöht (> 50 %)	*20 %*

gaben die Interview-Teilnehmer/-innen *SA1*, *MA1*, *VI1* und *SE2* als Grund für diese Erhöhung eine vollständigere Spezifikation an. Das heißt, der Mehraufwand resultiert teilweise aus der Schließung der Lücken in der Spezifikation und ist daher für die Schaffung einer vollständigen Spezifikation in jedem Fall nötig.

60 % der Interview-Teilnehmer/-innen stimmten darüber hinaus einer mittleren Erhöhung des Aufwandes zu. Jeweils 20 % ZR erhielten eine leichte Erhöhung sowie eine starke Erhöhung des Aufwandes bei der Spezifikation eines SoCs. Erhöht sich der Aufwand bei der Spezifikation durch die Anwendung der Modellierungsmethode um 40 %, würde dies eine Steigerung der Kosten, bezogen auf das zu Beginn des Abschnittes genannte Beispiel für die SoC-Entwicklung, von etwa 3 Mio. auf etwa 4 Mio. Euro bedeuten. Somit konnte, aus Sicht der Befragten, durch die Anwendung der in dieser Arbeit entwickelten formalisierten Modellierungsmethode für die Spezifikation die gestellte Anforderung **A5** nicht erfüllt werden. Die Steigerung des Aufwandes bei der Spezifikation resultiert jedoch dabei zum Teil aus der Steigerung der Vollständigkeit bei der Spezifikation und lässt sich somit als nötiger Mehraufwand verstehen. Darüber hinaus ist eine Steigerung des Aufwandes bei der Spezifikation vertretbar, lässt sich doch als Resultat daraus der Aufwand in anderen Bereichen und besonders über die gesamte SoC-Entwicklung hinweg reduzieren.

Tabelle 6.14 zeigt das Ergebnis der Befragung über den Einfluss der modellbasierten Entwicklungsmethode auf den Aufwand beim Entwurf des SoCs verglichen mit der aktuell angewendeten Arbeitsweise. Hierbei gaben alle Interview-Teilnehmer/-innen als Tendenz eine Verringerung des Aufwandes an.

Einer leichten Verringerung des Aufwandes stimmten dabei 50 % der Interview-Teilnehmer/-innen zu. Weitere 30 % ZR erhielt die Verringerung im mittleren Bereich und 20 % ZR die Verringerung im starken Bereich. Bei einer Reduzierung des Aufwandes beim Systementwurf um 35 % würden die Kosten für den Entwurf bezogen auf den zu Beginn des Abschnittes gewählten Mittelwert von etwa 5 Mio.

Tabelle 6.14 Fragestellung, Antwortmöglichkeiten und Ergebnis Frage 4.2

Fragestellung	Antwortmöglichkeit	ZR
Wie wirkt sich die modellbasierte Entwicklungsmethode auf den Aufwand beim Entwurf verglichen mit der aktuell angewendeten Arbeitsweise aus?	stark verringert (> 50 %)	*20 %*
	mittel bis stark verringert (25–50 %)	*30 %*
	leicht verringert (< 25 %)	*50 %*
	unverändert	*0 %*
	leicht erhöht (< 25 %)	*0 %*
	mittel bis stark erhöht (25–50 %)	*0 %*
	stark erhöht (> 50 %)	*0 %*

auf etwa 3 Mio. Euro gesenkt werden. Als Gründe für die Verringerung wurden besonders die Reduzierung von Implementierungsfehlern aufgrund einer eindeutigeren und vollständigeren Spezifikation und die Reduzierung des Aufwandes durch den modellgetriebenen Entwurf genannt. Somit konnte, aus Sicht der Befragten, durch die hier vorgestellte modellbasierte Entwicklungsmethode der Aufwand beim Systementwurf reduziert und somit die Anforderung **A7** erfüllt werden.

Als Nächstes wurden die Interview-Teilnehmer/-innen zum Einfluss der Methode auf den Aufwand in der Verifikation befragt. Das Ergebnis der Befragung ist in Tabelle 6.15 als Diagramm dargestellt.

Tabelle 6.15 Fragestellung, Antwortmöglichkeiten und Ergebnis Frage 4.3

Fragestellung	Antwortmöglichkeiten	ZR
Wie wirkt sich die modellbasierte Entwicklungsmethode auf den Aufwand bei der Verifikation verglichen mit der aktuell angewendeten Arbeitsweise aus?	stark verringert (> 50 %)	*0 %*
	mittel bis stark verringert (25–50 %)	*30 %*
	leicht verringert (< 25 %)	*40 %*
	unverändert	*0 %*
	leicht erhöht (< 25 %)	*30 %*
	mittel bis stark erhöht (25–50 %)	*0 %*
	stark erhöht (> 50 %)	*0 %*

Bei dem Einfluss der modellbasierten Entwicklungsmethode dieser Arbeit auf den Aufwand bei der Verifikation gingen die Antworten bereits in Bezug auf die Tendenzen auseinander. So antworteten sieben Interview-Teilnehmer/-innen, dass aufgrund der erhöhten Eindeutigkeit, Vollständigkeit und Verständlichkeit der Spezifikation die Arbeit der Verifikationsingenieure/-innen erleichtert würde und somit

der Aufwand zwischen leicht (40 % ZR) und mittel (30 % ZR) sinke. Eine Reduzierung des Aufwandes bei der Verifikation von 30 % würde, bezogen auf das zu Beginn dieses Abschnittes genannten Mittelwerts eines SoCs, eine Reduktion von 5 Mio. auf circa 3,5 Mio. Euro bedeuten.

Drei der Interview-Teilnehmer/-innen, darunter der Interview-Teilnehmer VI1, gaben an, dass sich aufgrund der umfangreicheren und vollständigeren Spezifikation der Aufwand für die Verifikation erhöhe, gleichzeitig jedoch auch die Qualität zunehme. Somit resultiert die unterschiedliche Einschätzung des Einflusses auf den Aufwand in erster Linie aus der jeweiligen Sichtweise der Befragten. Die Reduzierung des Aufwandes bei der Verifikation wurde als Anforderung **A8** an die in dieser Arbeit entwickelte Entwicklungsmethode gestellt und kann laut 70 % der Interview-Teilnehmer/-innen erfüllt werden.

Tabelle 6.16 zeigt zum Ende des Abschnittes das Ergebnis der Interviewevaluierung – bezogen auf den Einfluss der modellbasierten Entwicklungsmethode – hinsichtlich des Aufwands über den gesamten SoC-Entwicklungs-Flow gesehen, da hierbei auch der Einfluss auf den Aufwand durch Nacharbeiten, Fehlkommunikation und Behebung von Fehlern betrachtet wird, welcher erst zu einem späteren Zeitpunkt der Entwicklung auftritt. Alle zehn Interview-Teilnehmer/-innen gaben als Tendenz eine Reduzierung des Aufwandes an.

Tabelle 6.16 Fragestellung, Antwortmöglichkeiten und Ergebnis Frage 4.4

Fragestellung	**Antwortmöglichkeiten**	**ZR**
Wie wirkt sich die modellbasierte Entwicklungsmethode auf den Aufwand bezogen auf die gesamte SoC-Entwicklung verglichen mit der aktuell angewendeten Arbeitsweise aus?	stark verringert (> 50 %)	*20 %*
	mittel bis stark verringert (25–50 %)	*80 %*
	leicht verringert (< 25 %)	*0 %*
	unverändert	*0 %*
	leicht erhöht (< 25 %)	*0 %*
	mittel bis stark erhöht (25–50 %)	*0 %*
	stark erhöht (> 50 %)	*0 %*

Darüber hinaus fiel das Ergebnis der Befragungen sehr eindeutig aus. So stimmten 80 % der Interview-Teilnehmer/-innen einer mittel bis starken Verringerung des Aufwandes über den gesamten SoC-Entwicklungs-Flow hinweg betrachtet zu. Weitere 20 % ZR erhielt der Skalenwert „stark verringert“. Sollte sich dieses Ergebnis auch auf Dauer in der SoC-Entwicklung erreichen lassen, könnte sich durch die in dieser Arbeit vorgestellte modellbasierte Entwicklungsmethode eine Reduzierung des Aufwandes von circa 40 % einstellen. Dies entspräche, bezogen auf die zu

Beginn des Abschnittes angegebenen Mittelwerte für die SoC-Entwicklung, einer Reduktion der Kosten von 20 Mio. Euro auf etwa 12 Mio. Euro sowie eine Reduktion der Entwicklungsdauer von vier auf zweieinhalb Jahre. Vergleicht man diesen reduzierten Wert der Kosten für die gesamte SoC-Entwicklung aus der Beispielrechnung mit der Summe der reduzierten Werte der Bereiche Spezifikation, Entwurf und Verifikation, fällt auf, dass die Summe der Einzelwerte mit 12,5 Mio. bereits den Wert der gesamten Kosten übersteigt. Darüber hinaus ist zu beachten, dass in den zu Beginn des Abschnitts angenommenen Kosten für die gesamte SoC-Entwicklung (20 Mio. Euro) bereits die „Sonstigen Kosten“ (7 Mio. Euro) stellvertretend für die Bereiche Projektplanung, Fehlerhebung und Maskendruck enthalten sind.

Wird der Einfluss der modellbasierten Entwicklungsmethode auf die gesamten Kosten der SoC-Entwicklung betrachtet, resultiert die starke Senkung der Kosten, so *MA1,* neben der Reduktion der Kosten bei Entwurf und Verifikation besonders aus der Reduzierung der Kosten bei der Fehlerbehebung und aus der Vermeidung von Rekursionen bei der Maskenherstellung des Wafers. Muss diese Maskenerstellung aufgrund von Fehlern oder nachträglichen Anpassungen wiederholt werden, entsteht laut *MA1* eine Verzögerung von etwa einem Jahr und etwa 1 Mio. Euro an Mehrkosten pro Rekursion.

Dabei bleibt jedoch in diesem Abschnitt die Zunahme der Komplexität der zukünftig zu entwickelnden SoCs, welche der Reduktion des Aufwandes entgegenwirken würde, unbeachtet. Dennoch sehen, so das Ergebnis der Interviewevaluierung, alle Interview-Teilnehmer/-innen die Reduzierung des Aufwandes, bezogen auf die gesamte SoC-Entwicklung, durch Anwendung der modellbasierten Entwicklungsmethode als erfüllt an. Somit kann das in Kapitel 1 beschriebene übergeordnete Ziel der Effizienzsteigerung als erfüllt angesehen werden.

6.2 Evaluierung modellgetriebener Systementwurf

Wie bereits in der vorangegangen Arbeit [76] gezeigt wurde, kann bereits durch die teilweise Automatisierung des Entwurfs eine erhebliche Reduktion des Aufwandes erreicht werden. Dafür sind jedoch die Qualität und die Korrektheit des Generierungsergebnisses entscheidend. Aus diesem Grund wurde als Teil der hier vorliegenden Arbeit zusätzlich zu einer Evaluierung der Methode durch Interviews eine Verifikation des generierten Codes durchgeführt.

Co-Verifikation sowie Co-Simulation beschreiben die kombinierte Simulation von Hardware und Software. In der hier vorliegenden Arbeit wurde somit die

generierte Software nebenläufig zur generierten Hardware in Form des Virtuellen Prototyps simuliert und verifiziert. Der Virtuelle Prototyp dient somit bei der Simulation als Hardware-Emulator, welcher noch nicht der finalen Hardware des zu entwickelnden Systems entspricht, dabei jedoch das Verhalten der Hardware nachbildet. Für die Co-Simulation wurde dabei nachfolgend beschriebene Simulationsanordnung verwendet.

6.2.1 Simulationsanordnung

Für die Simulation wurde die in Abbildung 6.3 gezeigte Simulationsanordnung verwendet. Dabei zeigt die Abbildung 6.3, welche Tools für die Simulation des Virtuellen Prototyps in Verbindung mit der Software benötigt werden.

- Die zu simulierende Software wird mit **µVision Keil** kompiliert.
- Die kompilierte Software wird von dem in Abschnitt 5.3 beschriebenen Tool **Virtualizer Studio** geladen und ausgeführt.
- Das Debuggen der Software und die damit verbundene Steuerung der Simulation erfolgen mithilfe einer **Eclipse-„Integrated Development Environment**“ (IDE, Integrierte Entwicklungsumgebung).

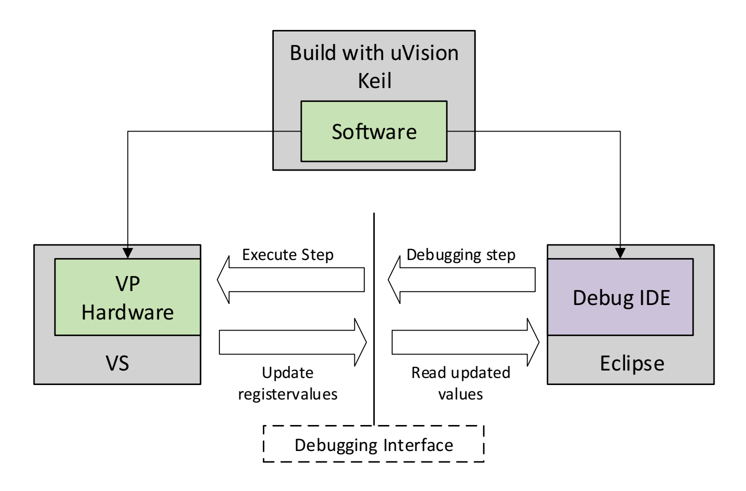


Abbildung 6.3 Simulationsanordnung Virtueller Prototyp

Dieser Ablauf gewährleistet, dass die Debugging Session und die SystemC-Simulation synchronisiert sind und das Debugging die Steuerung der Gesamtsimulation übernimmt. Während der laufenden Simulation können im Virtualizer Studio alle Registerwerte der Systemkomponenten und die Speicherinhalte ausgelesen und manipuliert werden.

6.2.2 Co-Verifikation

Die Co-Verifikation des generierten Codes erfolgt mittels dynamischen Testens und umfasst acht Testszenarien, welche bei den nachfolgenden Kombinationen zwischen Virtuellem Prototyp und Software angewandt wurden:

- Verifikation der generierten Software in Kombination mit manuell implementiertem und bereits verifiziertem Code des Virtuellen Prototyps
- Verifikation des generierten Codes des Virtuellen Prototyps in Kombination mit manuell implementiertem und verifiziertem Softwarecode
- Verifikation der Kombination aus generiertem Softwarecode und generiertem Code des Virtuellen Prototyps

Nachfolgend wird die Simulation der Testfälle anhand der Bootloading-Funktionalität beschrieben, welche Teil des in Abschnitt 5.2 vorgestellten *Processor-Subsystem* ist. Als Teil dieser Arbeit wurde die Bootloading-Funktionalität, wie in Abschnitt 5.2 erläutert, in SysML modelliert und der entsprechende C-Code mittels des in Abschnitt 5.3.3 beschriebenen Vorgehens generiert. Zusätzlich wurde die Architektur- wie auch die SystemC- Verhaltensbeschreibung der Hardware des Virtuellen Prototyps aus dem SysML-Modell generiert.

Der Bootloader ist das erste Programm, welches bei der Initialisierung eines SoCs ausgeführt wird. Eine der Hauptfunktionen des Bootloadings ist das Schreiben von Initialwerten aus dem *OTP* in die verschiedenen Register der Module durch das *CRC_module* beim Start des SoCs.

Bei der Durchführung der Co-Simulation wurden acht verschiedene Testszenarien ausgeführt. Jedes der Testszenarien **T1** bis **T8** beinhaltet einen definierten Initialwert im OTP (One-Time-Programmable), welcher zu Beginn der Simulation geladen werden muss. Diese Initialwerte werden bei der Ausführung des Bootloadings ausgelesen und an eine vordefinierte Adresse des Systems geschrieben. Die Adressen der unterschiedlichen Testszenarien sind sowohl RAM-Adressen als auch Konfigurationsregister einzelner Hardware-Komponenten. Das Bootloading

wird bei der Simulation jedes einzelnen Testszenarios vollständig ausgeführt, und anschließend werden die Registerwerte und der RAM ausgelesen. Die so ausgelesenen Werte und zugehörigen Adressen werden mit den Erwartungswerten aus der Testspezifikation der jeweiligen Testszenarien verglichen und verifiziert. Dieses Vorgehen entspricht dem White-Box-Prinzip [17].

Das Beispiel des Bootloadings ist in diesem Zusammenhang besonders aussagekräftig, da bei dessen Ausführung die kontrollflussorientierten Module des Virtuellen Prototyps von der Software angesteuert werden. Dabei wird somit die gesamte in Abbildung 5.11 gezeigte Architektur des Beispiel-SoCs wie auch die Software bei der Co-Simulation genutzt und getestet.

Nach der Verifikation des korrekten Verhaltens sowohl der generierten Software als auch des generierten Virtuellen Prototyps für das Bootloading wurde die Co-Simulation mit den Testszenarien **T1** bis **T8** ein weiteres Mal wiederholt, um das Verhalten der Software im Fehlerfall zu verifizieren. So müssen beispielsweise Fehler bei der Übertragung der Initialwerte durch das *CRC Module* mittels Redundanzcheck erkannt und korrigiert werden. Daher wurden zusätzliche Testfälle entwickelt, welche Fehler bei der Durchführung des Bootloadings provozieren und die Korrektheit des Verhaltens im Fehlerfall verifizieren. Dabei ist entscheidend, dass der fehlerhaft übertragene und anschließend geschriebene Initialwert im RAM beziehungsweise im Konfigurationsregister der Hardware-Komponenten dem zu erwartenden Wert der Testspezifikation entspricht.

Durch das hier beschriebene Vorgehen für die Co-Simulation mittels dynamischen Testens wurde versucht, durch die Testszenarien **T1** bis **T8** alle möglichen Verzweigungen der Software zu simulieren und zu prüfen, um so eine möglichst hohe Testabdeckung (Code Coverage) zu erreichen.

Testergebnisse

Als Ergebnis der in dieser Arbeit durchgeführten und in den vorangegangenen Abschnitten beschriebenen Tests lässt sich die Abwesenheit von Fehlern sowie die Abwesenheit von funktionellen Unterschieden zwischen generiertem und manuell implementiertem Code belegen. Dies gilt sowohl für die generierten Anteile der Software als auch für die in SystemC generierten kontrollflussorientierten Module des Virtuellen Prototyps. Zudem wurde ein korrektes Verhalten im Fehlerfall nachgewiesen. Jedoch geben die Tests keine Informationen über die Unterschiede in der Performance zwischen generiertem und manuell implementiertem Code.

6.2.3 Performance-Tests

Neben der Verifikation der korrekten Funktionalität der generierten Software und des generierten Virtuellen Prototyps im Vergleich zum manuell implementierten Code sollte die Performance des jeweiligen Codes verglichen werden. Dazu wurde im ersten Schritt die Laufzeit des manuell implementierten Codes und des generierten Codes bei der Ausführung der im vorangegangenen Abschnitt beschriebenen Testszenarien **T1** bis **T8** gemessen und verglichen. Tabelle 6.17 zeigt das Ergebnis der Messung.

Tabelle 6.17 Performance-Test: Laufzeit

Testszenarien	**Laufzeit**		
	Manuell Implementierter Code [µs]	***Generierter Code [µs]***	***Abweichung in %***
Testszenario T1	37,262	37,223	0,001
Testszenario T2	37,457	37,340	0,003
Testszenario T3	24,410	24,293	0,005
Testszenario T4	19,122	19,396	−0,014
Testszenario T5	19,982	20,060	−0,004
Testszenario T6	21,085	21,202	−0,006
Testszenario T7	21,124	21,202	−0,004
Testszenario T8	30,480	30,519	−0,001

Die Tabelle 6.17 zeigt die ermittelte Laufzeit des manuell implementierten Codes im Vergleich zum generierten Code, gemessen in Mikrosekunden. In der vierten Spalte wurde die Differenz zwischen den beiden Werten berechnet und prozentual dargestellt. Die gemessenen Laufzeiten weisen dabei nur minimale Unterschiede auf. Diese lassen sich mit hoher Wahrscheinlichkeit auf andere Parameter zurückführen, welche zu üblichen Schwankungen bei der Simulationsgeschwindigkeit führen können. Somit lässt sich für die generierten Funktionen, wie zu erwarten, eine vergleichbare Performance bezogen auf die Laufzeit nachweisen.

Zusätzlich zur Laufzeit wurde im nächsten Schritt der Speicherbedarf der Bootloading-Teilfunktionen im ROM gemessen und verglichen. Die jeweilige Software-Funktion wurde dazu kompiliert und der ROM-Verbrauch der drei

internen Bootloading-Funktionen ausgelesen und verglichen. Das Ergebnis des Vergleichs wird in Tabelle 6.18 veranschaulicht.

Tabelle 6.18 Performance-Test: Speicherbedarf

Bootloading Interne Funktion	**ROM-Speicherbedarf**		
	Manuell Implementierter Code [Byte]	***Generierter Code [Byte]***	***Abweichung in %***
Teilfunktion TF1	107,42	107,42	0,00 %
Teilfunktion TF2	642,58	636,72	0,91 %
Teilfunktion TF3	332,03	330,07	0,59 %

Die Tabelle 6.18 zeigt den ermittelten Speicherverbrauch des manuell implementierten Codes der drei Bootloading-Teilfunktionen verglichen mit dem jeweiligen generierten Code in Byte. In der vierten Spalte ist die berechnete prozentuale Abweichung dargestellt. Bei der Funktion 1 wurde, wie in der Tabelle 6.18 zu sehen ist, kein Unterschied beim Speicherverbrauch gemessen. Im Gegensatz dazu zeigen die Funktionen 2 und 3 leichte Unterschiede. Dies lässt sich auf kleinere Unterschiede im generierten Code verglichen mit dem implementierten Code zurückführen, welche die Funktionalität des Codes jedoch nicht beeinflussen. So wurde beispielsweise bei der manuellen Implementierung eine Switch-Case-Anweisung implementiert, während der Generator dasselbe Verhalten als If-Else-If-Verkettung generiert. Interessant ist jedoch, dass die generierten Teilfunktionen 2 und Teilfunktion 3 einen geringeren Wert für den Speicherverbrauch als der manuell implementierte Code aufweisen. Da die Unterschiede jedoch lediglich gering sind, lässt sich auch hier – verglichen mit dem manuell implementierten Code – eine vergleichbare Performance bezogen auf den Speicherverbrauch des generierten Codes nachweisen.

6.3 Zwischenergebnis

In dem vorangegangenen Kapitel konnte, mittels Interviewevaluierung, die Reduzierung der diskutierten Defizite durch die Anwendung der in dieser Arbeit entwickelten modellbasierten Entwicklungsmethode evaluiert werden. Dabei wurden neben der Analyse der Ursachen für das jeweilige Defizit mögliche Lösungsansätze für die Reduzierung des Defizites analysiert. Es konnte somit nachgewiesen

werden, dass auf Basis der Erfahrungen der Befragten die in dieser Arbeit vorgestellte modellbasierte Entwicklungsmethode die priorisierten Lösungsansätze für die Reduktion des Defizites enthält und somit in der Lage ist, die Effizienz der Entwicklung zu erhöhen. Darüber hinaus wurden die Interview-Teilnehmer/-innen zu dem Einfluss der modellbasierten Entwicklungsmethode auf den Aufwand in den Bereichen Spezifikation, Entwurf, Verifikation und in dem gesamten SoC-Entwicklungs-Flow befragt. Hierbei konnte festgestellt werden, dass zwar der Aufwand bei der Spezifikation steigt, der Aufwand der anderen Bereiche und der gesamten SoC-Entwicklung durch die Reduzierung von auftretenden Defiziten und die Verbesserung der Methodik jedoch erheblichen verringert werden kann.

Neben der Evaluierung der Methode wurde durch die Co-Verifikation und die Durchführung verschiedener Performance-Tests die Qualität und Korrektheit des generierten Codes sowohl im Bereich der Software als auch im Bereich des Virtuellen Prototyps nachgewiesen.

In Kapitel 7 folgt die Zusammenfassung der Arbeit sowie ein Ausblick möglicher zukünftiger Themen für weiterführende Arbeiten auf dem Gebiet der modellbasierten Entwicklungsmethoden.

7 Zusammenfassung und Ausblick

Im folgenden Kapitel wird die hier vorliegende Arbeit zusammengefasst. Dazu wird auf die in dieser Arbeit vorgestellte modellbasierte Entwicklungsmethode eingegangen. Zudem werden die daraus resultierenden Ergebnisse in gekürzter Form dargestellt. Anschließend folgt ein Ausblick auf mögliche weiterführende Arbeiten, um die Effizienz der SoC-Entwicklung in Zukunft weiter zu optimieren.

7.1 Zusammenfassung

Aufgrund der zu erwartenden weiter steigenden Komplexität zukünftiger Automotive SoCs werden die Herausforderungen für die Spezifikation, den Entwurf und die Verifikation weiter steigen. Um dennoch zukünftig effizient und wirtschaftlich zu bleiben, müssen neue Methoden gefunden werden. Ferner muss die Automatisierung auch im Bereich der Entwicklung vorangetrieben werden.

Im Zuge der Arbeit wurde dazu eine modellbasierte Entwicklungsmethode und als Teil dessen eine Methode zur abstrakten Modellierung von Automotive Systems-on-Chips vorgestellt. Das übergeordnete Ziel der Arbeit war dabei, die Steigerung der Effizienz in der SoC-Entwicklung zu erreichen. Die modellbasierte Entwicklungsmethode dieser Arbeit setzt sich aus mehreren Teilmethoden wie folgt zusammen:

Die Entwicklung einer Modellierungsmethode für die modellbasierte Systemkonzeptentwicklung und die damit verbundene Implementierung einer Arbeitsweise nach dem Top-down-Ansatz zielt auf die Minimierung der Defizite bei der Entwicklung des Systemkonzepts ab. Darüber hinaus behandelt die Arbeit die Formalisierung und Modellierung der Spezifikation über den gesamten SoC-Entwicklungs-Flow hinweg, um so die Eindeutigkeit der spezifizierten Informationen zu erhöhen. Dabei beinhaltet die Modellierungsmethode einen

A. Kirchner, *Entwicklung von Methoden zur abstrakten Modellierung von Automotive Systems-on-Chips*, https://doi.org/10.1007/978-3-658-38437-1_7

kombinierten Ansatz aus Systemanforderungen in natürlicher Sprache und formalisierter Modellierung des Systems zur Spezifikation und beschäftigt sich darüber hinaus mit der Integration der Methode in den aktuellen SoC-Entwicklungs-Flow. Durch die Steigerung der Konsistenz, der Fehlerfreiheit und der Eindeutigkeit der Spezifikation kann indirekt eine Aufwandsreduzierung erzielt werden, welche auf der Senkung von benötigten Nacharbeiten und Fehlerbehebungen beruht.

Darauf aufbauend wurde ein Verfeinerungsprozess für modellbasierte Spezifikationen als Teil des Übergangs zwischen Konzeptphase und Entwurfsphase vorgestellt. Dabei wird das in der Konzeptphase auf System-Ebene entwickelte Systemkonzept durch implementierungsnahe Elemente der Spezifikation, wie z. B. der Verhaltensbeschreibung der einzelnen Hard- und Software Module erweitert.

Zusätzlich beschäftigt sich die Arbeit mit Ansätzen der modellgetriebenen Automatisierung des Systementwurfs, um den Entwicklungsaufwand unmittelbar zu reduzieren. Dabei wurde die Generierung von Virtuellen Prototypen behandelt. Die Verwendung von Virtuellen Prototypen ermöglicht eine frühere Entwicklung der On-Chip-Software und reduziert die Fehlerhäufigkeit in späten Phasen der Entwicklung. Mit dem Ziel, den manuellen Aufwand bei der Entwicklung von Virtuellen Prototypen zu reduzieren, baut die Arbeit auf eine bestehende Lösung aus [15] auf, welche sich in erster Linie auf die Generierung der signalverarbeitenden Anteile des SoCs auf Grundlage von MATLAB-Simulink-Modellen konzentriert. Dazu wurde durch Anwendung der in der vorliegenden Arbeit gezeigten modellgetriebenen Entwicklungsmethode die bestehende Lösung um Lösungen für die modellbasierte Generierung der kontrollflussorientierten Anteile des SoCs sowie der Architektur erweitert. Durch die Kombination der Lösungen kann die nahezu vollständige Generierung eines Virtuellen Prototyps ermöglicht werden.

Darauf aufbauend wurde die Generierung der On-Chip-Software als Teil des Systementwurfs in der hier vorliegenden Arbeit ermöglicht. Durch diesen Schritt konnte der Aufwand für den Systementwurf selbst gesenkt werden und so die Effizienz der SoC-Entwicklung gesteigert werden. Ein wichtiger Nebeneffekt bei der Generierung von Code auf Basis der modellierten Spezifikation ist die zusätzliche Senkung der Fehlerwahrscheinlichkeit sowie die Steigerung der Konsistenz zwischen Spezifikation und implementiertem Entwurf.

Neben dem Entwurf eines Systems wird ein stetig wachsender Anteil des Entwicklungsaufwandes für die Verifikation der immer komplexer werdenden Systeme benötigt. Daher wurde in dieser Arbeit zusätzlich die Automatisierung der Verifikationserstellung behandelt. Dazu wurden die benötigten Informationen für die Generierung aus Architektur-Diagrammen des Systemmodells extrahiert

und anschließend auf dessen Basis die Generierung von SystemVerilog-Assertions für die Implementierung von „Connectivity Checks" ermöglicht. Neben der enormen Wichtigkeit einer vollständigen und eindeutigen Spezifikation, welche in dieser Arbeit angegangen wurde, konnte durch die Automatisierung ein erster Schritt in Richtung Reduzierung des Verifikationsaufwandes ermöglicht werden.

Die Evaluierung der Methode und deren Einfluss auf die Effizienz in der SoC-Entwicklung wurde anhand einer Interviewevaluierung durchgeführt. Dabei wurden zehn Interview-Teilnehmer/-innen aus dem Bereich der SoC-Entwicklung, welche bei der Anwendung der Methode beteiligt waren, befragt, um so eine Aussage über den Einfluss der Methode auf die Effizienz der SoC-Entwicklung treffen zu können.

Im Rahmen der Evaluierung der Methode wurde zudem der generierte Code des Softwareentwurfs und des Virtuellen Prototyps mittels Tests und Simulation auf Fehlerfreiheit und Korrektheit getestet. Hierdurch konnte gezeigt werden, dass der generierte Code dem manuell implementierten Code gleichwertig ist.

7.2 Ausblick

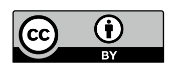
Zusätzlich zu der in dieser Arbeit vorgestellten modellbasierten Entwicklungsmethode beinhaltet die Arbeit ein „Proof-of-Concept" der vorgestellten Lösungen für die modellbasierte Automatisierung des SoC-Entwurfs sowie der Verifikationserstellung. Dadurch konnte eine erste Aussage über das Potenzial geboten werden, welches durch eine Modellierung und Formalisierung der Spezifikation erreicht werden kann. Die Formalisierung aller Teile der Spezifikation sowie die Schaffung einer Spezifikation, bestehend aus verschiedenen vollverlinkten Beschreibungen in verschiedenen Formaten, bedarf zukünftiger Arbeiten. Darüber hinaus bietet die Modellierung des Systems und damit die Beschreibung in einer maschinenlesbaren Sprache weiteres Potenzial für die automatisierte Interaktion zwischen Spezifikation und weiterführenden EDA-Tools.

Wie in Abschnitt 6.1 durch das Evaluierungsinterview deutlich wurde, ist die in dieser Arbeit noch nicht vollumfänglich erfüllte Simulierbarkeit der Spezifikation entscheidend für eine erfolgreiche Kommunikation mit dem Kunden und könnte daher in einer weiterführenden Arbeit realisiert werden.

Das in Abschnitt 4.1 vorgestellte Automotive-SoC-Profil wurde auf Basis von Erfahrungswerten, „Stakeholder"-Befragungen und der Arbeit an dem Pilotprojekt entwickelt und wird auch zukünftig weiterführende Arbeiten benötigen, um es an die jeweiligen Bedürfnisse der SoC-Entwicklung anzupassen.

Darüber hinaus besteht ein großes Potenzial in der Weiterentwicklung der hier vorgestellten Lösungen zur modellbasierten Automatisierung in der SoC-Entwicklung. Als weiterer Schritt der Automatisierung – und damit Grundlage für eine weiterführende Arbeit – könnte beispielsweise die Generierung von VHDL-Code auf Basis des SysML-Systemmodells ermöglicht werden. Die VHDL ist eine Hardwarebeschreibungssprache, die in nahezu allen SoC-Entwicklungen eingesetzt wird. Die Generierung der Hardware-Architektur in VHDL auf Basis einer SysML-Architektur-Beschreibung birgt neben der enormen Reduzierung des Entwurfsaufwandes große Potenziale im Bereich der Konsistenz zwischen Spezifikation und Implementierung sowie der Reduzierung von Implementierungsfehlern. Zwar gewinnt die Software in heutigen SoC-Entwicklungen weiter an Bedeutung, jedoch wird auch heute noch ein Großteil der Funktionen aufgrund der Forderungen nach Echtzeitverhalten mittels Hardware realisiert. Ließe sich eine modellgetriebenen VHDL-Generierung für den Hardware-Entwurf im Bereich der SoC-Entwicklung realisieren, wäre damit die Basis dafür geschaffen, gesamte Subsysteme eines Automotive SoCs auf Basis der modellierten Spezifikation vollautomatisch zu generieren.

Auch die Automatisierung der Verifikationserstellung ist ein entscheidendes und zukünftig wichtiges Themenfeld. Neben einer Steigerung der Spezifikationsqualität birgt die Automatisierung der Verifikationserstellung ein erhebliches Potenzial, um den Aufwand zu senken und so die Effizienz zu erhöhen.



Literaturverzeichnis

1. F. P. Brooks, The Mythical Man-Month, Amsterdam: Addison-Wesley Longman, 1995.
2. C. Haubelt und J. Teich, „Einleitung," in *Digitale Hardware/Software-Systeme: Spezifikation und Verifikation*, Heidelberg, Springer-Verlag, 2010, pp. 1–22.
3. F. Müller und J. Pöppelbuß, „Rollenverschiebungen in der Automobilentwicklung," Universität Bremen, Bremen, 2015.
4. H. Wallentowitz, A. Freialdenhoven und I. Olschewski, in *Strategien in der Automobilindustrie: Technologietrends und Marktentwicklungen*, Wiesbaden, Vieweg+Teubner, 2009, pp. 1–27.
5. F.-R. Esch, „Herausforderungen der Automobilindustrie entgegentreten," in *Strategie und Technik des Automobilmarketing*, Wiesbaden, Springer Gabler, 2013, pp. 25–30.
6. GP Bullhound, „Verteilung des Umsatzes in der weltweiten Automobilindustrie nach Geschäftsbereichen in den Jahren 2015 und 2030," Statista GmbH, 2019. [Online]. Available: https://de.statista.com/statistik/daten/studie/1022441/umfrage/verteilung-des-umsatzes-in-der-weltweiten-automobilindustrie-nach-geschaeftsbereichen/. [Zugriff am Mai 2020].
7. F.-R. Esch, E. von Einem und V. Rühl, „Bedeutung und Rahmenbedingungen der Segmentierung in der Automobilbranche," in *Strategie und Technik des Automobilmarketing*, Wiesbaden, Springer Gabler, 2013, pp. 63–64.
8. Handelsblatt, „Pkw-Dichte in ausgewählten weltweiten Ländern im Jahr 2015 (Pkw je 1.000 Einwohner)," Statista GmbH, 2016. [Online]. Available: https://de.statista.com/statistik/daten/studie/163407/umfrage/pkw-dichte-in-ausgewaehlten-laendern/. [Zugriff am Mai 2020].
9. Verband der Automobilindustrie, „Jahresbericht 2018," 16 November 2018. [Online]. Available: http://www.vda.de. [Zugriff am 21 Februar 2020].
10. F.-R. Esch, E. von Einem und C. Kuklinski, „Markendehnung als Wachstumstreiber der Automobilindustrie," in *Strategie und Technik des Automobilmarketing*, Wiesbaden, Springer Gabler, 2013, pp. 361–363.
11. O. Bringmann, W. Lange und M. Bogdan, „Einführung, Bauformen und Technologien," in *Eingebettete Systeme: Entwurf, Modellierung und Synthese*, Berlin/Boston, Walter de Gruyter GmbH, 2018, pp. 1–26.
12. E. Hering, K. Bressler und J. Gutekunst, Elektronik für Ingenieure und Naturwissenschaftler, Berlin, Heidelberg: Springer Verlag, 2017.

A. Kirchner, *Entwicklung von Methoden zur abstrakten Modellierung von Automotive Systems-on-Chips*, https://doi.org/10.1007/978-3-658-38437-1

13. ISO 26262-8:2011, „Road vehicles: Functional safety,“ International Organization for Standardization, 2011.
14. O. Bringmann, W. Lange und M. Bogdan, „Entwicklungsmethodiken,“ in *Eingebettete Systeme: Entwurf, Modellierung und Synthese*, Berlin/Boston, Walter de Gruyter GmbH, 2018, pp. 27–70.
15. A. Mauderer, A. Schreiber, J. Chevalier und J.-H. Oetjens, „Automated Integration of MathWorks® Simulink® Signal Flow Graph Models into Synopsys® Virtualizer TM-based Virtual Prototypes,“ Synopsys User Group Conference 2016, 2016.
16. G. Pahl, W. Beitz, J. Feldhusen und K.-H. Grote, Pahl/Beitz Konstruktionslehre: Grundlagen erfolgreicher Produktentwicklung. Methoden und Anwendung, Heidelberg: Springer-Verlag, 2007, pp. 194–198.
17. O. Bringmann, W. Lange und M. Bogdan, „Verifikation, Simulation und Test,“ in *Eingebettete Systeme: Entwurf, Modellierung und Synthese*, Berlin/Boston, Walter de Gruyter GmbH, 2018, pp. 295–312.
18. V. Gebhardt, G. M. Rieger, J. Mottok und C. Gießelbach, „Verifikations- und Validationsplanung,“ in *Funktionale Sicherheit nach ISO 26262: Ein Praxisleitfaden zur Umsetzung*, Heidelberg, dpunkt.verlag, 2013, pp. 135–166.
19. G. Liebel, M. Tichy, E. Knauss, O. Ljungkrant und G. Stieglbauer, „Organisation and communication problems in automotive requirements engineering,“ Springer, 2016.
20. D. Dori, „PART I Model-Based Systems Engineering Introduced,“ in *Model-Based Systems Engineering with OPM and SysML*, New York, Springer Science+Business Media, 2016, pp. 3–35.
21. E. Barke, M. Olbrich, L. Hedrich, V. Burkhay und M. Freisfeld, „Electronic Design Automation (EDA): Entwurfsautomatisierung in der Mikroelektronik,“ IMS Leibniz Universität Hannover, [Online]. Available: http://edascript.ims.uni-hannover.de/de/index.html. [Zugriff am Juni 2020].
22. T. Weilkiens, Systems Engineering mit SysML/UML, Heidelberg: dpunkt.verlag, 2016, pp. 157–213.
23. OMG, „OMG Unified Modeling Language TM (OMG UML),“ März 2015. [Online]. Available: https://www.omg.org/spec/UML/2.5/PDF. [Zugriff am Mai 2020].
24. OMG, „UML Profile for MARTE,“ April 2019. [Online]. Available: https://www.omg.org/spec/MARTE/. [Zugriff am April 2020].
25. OMG, „OMG System Modeling Language,“ Mai 2017. [Online]. Available: https://www.omg.org/spec/SysML/1.5/. [Zugriff am Juni 2020].
26. U. Frank, „Domain-Specific Modeling Languages: Requirements Analysis and Design Guidelines,“ in *Domain Engineering: Product Lines, Languages and Conceptual Models*, Heidelberg, Springer Verlag, 2013, pp. 134–140.
27. D. D. Gajski und R. Kuhn, Guest Editor’s Introduction: New VLSI Tools, IEEE Computer, 1983.
28. IEEE Computer Society, „IEEE Std 1666–2005, Standard SystemC Language Reference Manual,“ 2012.
29. F. Kesel, Modellierung von digitalen Systemen mit SystemC: Von der RTL-zur Transaction-Level-Modellierung, Munich, Germany: Oldenbourg Verlag, 2012.
30. IEEE, IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology, 1990.
31. J.-A. Maxa, Model Driven Development for Embedded Software, Elsevier, 2018.

32. OMG, „MDA Guide revision 2.0," Juni 2014. [Online]. Available: https://www.omg.org/cgi-bin/doc?ormsc/14-06-01. [Zugriff am Mai 2020].
33. S. Rigo, R. Azevedo und L. Santos, Electronic System Level Design, Heidelberg: Springer Verlag, 2011.
34. K. Pohl, H. Hönninger, R. Achatz und M. Broy, Model-Based Engineering of Embedded Systems The SPES 2020 Methodology, Heidelberg: Springer Verlag, 2012.
35. Z. Karakehayov, „Hierarchical design model for embedded systems," 2009 IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications , Rende, Italy, 2009 .
36. A. Basu und S. Bhattacharya, „A Formal Specifikation language for domain specific software development," 2004 IEEE Region 10 Conference TENCON , 2004.
37. P. Zhang, „Specification and verification of SOC using PTL," 2009 Asia-Pacific Conference on Computational Intelligence and Industrial Applications (PACIIA), Wuhan, China, 2009.
38. A. Fatwanto, „Translating software requirements from natural language to formal specification," 2012 IEEE International Conference on Computational Intelligence and Cybernetics (CyberneticsCom), Bali, Indonesia, 2012.
39. T. Schattkowsky und W. Mueller, „Model-based specification and execution of embedded real-time systems," Proceedings Design, Automation and Test in Europe Conference and Exhibition, Paris, France, 2004.
40. T. Schattkowsky und W. Müller, „Model-based design of embedded systems," Seventh IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2004. Proceedings, Vienna, Austria, 2004.
41. . T. Schattkowsky, W. Mueller und A. Rettberg , „A model-based approach for executable specifications on reconfigurable hardware," Design, Automation and Test in Europe, Munich, Germany, 2005.
42. L. Musat, S. Kandl, P. Puschner, M. Hübl, A. Buzo und G. Pelz, „Requirement semi-formalization methodology for SoC design," 2015 International SoC Design Conference (ISOCC), Gyungju, South Korea, 2015.
43. . L. S. Indrusiak und M. Glesner, „SoC Specification using UML and Actor-Oriented Modeling," 2006 International Biennial Baltic Electronics Conference, Tallinn, Estonia, 2006.
44. C. F. G. Ribeiro, A. Rettbergy, C. E. Pereira, C. Steinmetz und M. S. Soares, „SPES Methodology and MARTE Constraints in Architectural Design," 2018 IEEE Symposium on Computers and Communications (ISCC), Natal, Brazil, 2018.
45. F. G. C. Ribeiro, C. E. Pereira, A. Rettberg und M. S. Soares, „Model-based requirements specification of real-time systemswith UML, SysML and MARTE," in *Software & Systems Modeling*, Heidelberg, 2018, p. 343–361.
46. F. Herrera, J. Medina und E. Villar, „Modeling Hardware/Software Embedded Systems with UML/MARTE: A Single-Source Design Approach," in *Handbook of Hardware/Software Codesign*, Dordrecht, Springer Science+Business Media, 2017, pp. 141–185.
47. Y. W. Hau, M. Khalil-Hani und M. N. Marsono, „CODESL:A Framework for System-level Modelling, Co-simulation and Design-Space Exploration of Embedded Systems based on System-on-Chip," 2010 International Conference on Intelligent Systems, Modelling and Simulation, Liverpool, UK, 2010.

48. T. Schattkowsky, T. Xie und W. Mueller, „A UML frontend for IP-XACT-based IP management," Design, Automation & Test in Europe Conference & Exhibition, Nice, France, 2009.
49. M. d. S. Soares und J. Vrancken, „Requirements specification and modeling through SysML," 2007 IEEE International Conference on Systems, Man and Cybernetics, Montreal, Que., Canada, 2007.
50. M. R. S. Marques, E. Siegert und L. Brisolara, „Integrating UML, MARTE and sysml to improve requirements specification and traceability in the embedded domain," 2014 12th IEEE International Conference on Industrial Informatics (INDIN), Porto Alegre, Brazil, 2014 .
51. T. S. Costa, A. Sampaio und G. Alves, „Using SysML in Systems Design," 2009 International Conference on Information Management, Innovation Management and Industrial Engineering, Xi'an, China , 2009 .
52. Y. Vanderperren und W. Dehaene, „UML 2 and SysML: an approach to deal with complexity in SoC/NoC design," Design, Automation and Test in Europe, Munich, Germany, 2005.
53. R. Kawahara, D. Dotan, T. Sakairi, K. Ono und H. Nakamura, „Verification of embedded system's specification using collaborative simulation of SysML and simulink models," 2009 International Conference on Model-Based Systems Engineering, Haifa, Israel, 2009 .
54. The MathWorks Inc., „MATLAB," 2020. [Online]. Available: https://www.mathworks.com/products/matlab.html. [Zugriff am Juni 2020].
55. The MathWorks Inc., „Simulink," 2020. [Online]. Available: http://www.mathworks.com/products/simulink/. [Zugriff am Juni 2020].
56. Sparx Systems, „Enterprise Architect 15.1 User Guide," 2020. [Online]. Available: https://www.sparxsystems.com/enterprise_architect_user_guide/15.1/index/index.html. [Zugriff am Juli 2020].
57. Mirabilis Design, „VisualSim Architect," 2020. [Online]. Available: https://www.mirabilisdesign.com/visualsim-architect/. [Zugriff am Juli 2020].
58. Eclipse Foundation, „Papyrus User Guide," 2020. [Online]. Available: https://wiki.eclipse.org/Papyrus_User_Guide. [Zugriff am Juli 2020].
59. IBM, „IBM Rational Rhapsody User Guide," 2019. [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SSB2MU_8.4.0/com.ibm.rhp.homepage.doc/helpindex_rhapsody.html. [Zugriff am Juni 2020].
60. W. Mueller, D. He , F. Mischkalla, A. Wegele , P. Whiston und P. Peñil, „The SATURN Approach to SysML-Based HW/SW Codesign," 2010 IEEE Computer Society Annual Symposium on VLSI, Lixouri, Kefalonia, Greece, 2010 .
61. F. Mischkalla, D. He und W. Mueller, „Closing the gap between UML-based modeling, simulation and synthesis of combined HW/SW systems," 2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010), Dresden, Germany, 2010.
62. IT Central Station, „Atego Artisan Studio," Juli 2020. [Online]. Available: https://www.itcentralstation.com/products/atego-artisan-studio-reviews. [Zugriff am Juli 2020].
63. PTC, „PTC Integrity Modeler," 2020. [Online]. Available: https://www.ptc.com/de/products/windchill/integrity/integrity-modeler-what-is- new. [Zugriff am Juli 2020].

64. . K. Nguyen, Z. Sun, P. Thiagarajan und W.-F. Wong , „Model-driven SoC design via executable UML to SystemC," 25th IEEE International Real-Time Systems Symposium, Lisbon, Portugal, 2004.
65. The Apache Software Foundation, „Apache Verlocity Engine 2.0 User Guide," [Online]. Available: velocity.apache.org/engine/2.0/user-guide.html. [Zugriff am Juli 2020].
66. R. Görgen und E. d. Kock, „SysML based Architecture Definition and Platform Generation Flow," 2019 Design and Verifikation Conference and Exhibition (DVCON), München, 2019.
67. M. Baklouti, M. Ammar, P. Marquet, M. Abid und J.-L. Dekeyser, „A model-driven based framework for rapid parallel SoC FPGA prototyping," 2011 22nd IEEE International Symposium on Rapid System Prototyping, Karlsruhe, 2011.
68. Modelica Association, „Modelica® -A Unified Object-Oriented Language for Systems Modeling," 10 April 2017. [Online]. Available: https://www.modelica.org/documents/ModelicaSpec34.pdf. [Zugriff am Juli 2020].
69. Open Source Modelica Consortium, „OpenModelica User's GuideRelease v1.16," 31 Juli 2020. [Online]. Available: https://www.openmodelica.org/useresresources/userdocumentation. [Zugriff am Juli 2020].
70. Synopsys, „Virtualizer Studio User's Guide," 2016. [Online]. Available: https://www.synopsys.com/verification/virtual-prototyping/virtualizer.html. [Zugriff am Juni 2020].
71. Synopsys, „Platform Architect Ultra," 2020. [Online]. Available: https://www.synopsys.com/verification/virtual-prototyping/platform-architect.html. [Zugriff am Juli 2020].
72. Menthor Graphics, „Vista Architect Datasheet," 2020. [Online]. Available: https://s3.amazonaws.com/s3.mentor.com/public_documents/datasheet/esl/vista/vista-architect-ds.pdf. [Zugriff am Juli 2020].
73. Cadence Design System, „Virtual System Platform," 2011. [Online]. Available: https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/Archive/virtual_system_platform_ds.pdf. [Zugriff am Juli 2020].
74. R. Baranowski und M. Trunzer, „Complete Formal Verification of a Family of Automotive DSPs," 2016 Design and Verifikation Conference and Exhibition (DVCON), München, 2016.
75. U. Kühne, S. Beyer, J. Bormann und J. Barstow, „Automated formal verification of processors based on architectural models," Formal Methods in Computer Aided Design, Lugano, Switzerland, 2010.
76. A. Kirchner, J.-H. Oetjens und O. Bringmann, „Using SysML for Modelling and Code Generation for Smart Sensor ASICs," 2018 Forum on Specification & Design Languages (FDL), Garching, Germany, 2018.
77. A. Kirchner, J.-H. Oetjens und O. Bringmann, „Automation of Embedded Software Development for Smart Sensor ASICs," International Workshop on Embedded Software for the Industrial IOT (ESIIT) on Design, Automation and Test in Europe Conference 2019 (DATE 2019), Florence, Italy, 2019.
78. A. Kirchner, J.-H. Oetjens, T. Kogel und O. Bringmann, „Using SysML for Modeling and Generation of Virtual Platforms," Synopsys Users Group Europe, München, 2019.
79. A. Kirchner, J.-H. Oetjens und O. Bringmann, „Model-based Automation of Verification Development for automotive SOCs," 2020 Design and Verifikation Conference and Exhibition (DVCON), München, 2020.
80. KONZEPT . GRAFIK . DESIGN | TANJA KIRCHNER, *tree swing,* Stuttgart, 2022.

81. T. Niebisch, Anforderungsmanagement in Sieben Tagen : Der Weg Vom Wunsch Zur Konzeption, Wiesbaden: Springer Gabler, 2013.
82. Developers of various domains of Robert Bosch GmbH, Interviewee, *Personal Stakeholder-Interviews: "Domain-specific requirements for model-based system development,"*. [Interview]. September 2019.
83. „1685–2014 – IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows," IEEE Std 1685–2014 (Revision of IEEE Std 1685–2009) , vol., no., pp. 1–510, 2014.
84. A. Evans, A. Silburt, G. Vrckovnik, T. Brown und M. Dufresne, „Functional verification of large ASICs," Proceedings 1998 Design and Automation Conference. 35th DAC, San Francisco, CA, USA, 1998.
85. M. Luisa, F. Mariangela und N. I. Pierluigi , „Market research for requirements analysis using linguistic tools," in *Requirements Engineering volume 9*, Springer Link, 2004, p. 40–56.
86. J. Ruf, R. J. Weiss, T. Kropf und W. Rosenstiel, „Modeling and Formal Verification of Production Automation Systems," in *Integration of Software Specification Techniques for Applications in Engineering*, Heidelberg, Springer Verlag, 2004, pp. 541–566.
87. IEEE Std 1800–2017 , „IEEE Standard for SystemVerilog--Unified Hardware Design, Specification, and Verification Language," IEEE, 2013.
88. IBM, „IBM Rational DOORS and Rational DOORS Web Access V9.6.1 documentation," 2019. [Online]. Available: https://www.ibm.com/support/knowledgecenter/SSYQBZ_9.6.1/com.ibm.doors.requirements.doc/helpindex_ doors.html. [Zugriff am Juni 2020].
89. OASIS Open Project, „Open Services for Lifecycle Collaboration," [Online]. Available: https://open-services.net. [Zugriff am Juni 2020].
90. J. Sobolewski, „Cyclic redundancy check," in *4th Encyclopedia of Computer Science*, Nature Publishing Group, 2003, pp. 476–479.
91. Synopsys, „Virtualizer Studio Python Interface Reference Manual," 2016. [Online]. Available: https://www.synopsys.com/verification/virtual-prototyping/virtualizer.html. [Zugriff am Mai 2020].
92. Python Software Foundation, „28.1. sys — System-specific parameters and functions," 2019. [Online]. Available: https://docs.python.org/2/library/sys.html. [Zugriff am Juni 2020].
93. P. Mayring, Qualitative Content Analysis: Theoretical Foundation, Basic Procedures and Software Solution, Klagenfurt, Austria: gesis, 2014.
94. N. Menold und K. Bogner, „Gestaltung von Ratingskalen in Fragebögen," 2015. [Online]. Available: https://doi.org/10.15465/gesis-sg_015. [Zugriff am Januar 2021].